UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

PARALELIZACÓN Y OPTIMIZACIÓN DE UN CÓDIGO DE SIMULACIÓN DE IMÁGENES STM EN PLATAFORMAS MULTICORE

Luis Ciruelo Sanz
Tutor: Pablo Pou Bell
Ponente: Iván González Martínez

Julio 2015

Resumen

En la actualidad la nanotecnología y sus aplicaciones están cada vez más presentes en nuestra vida cotidiana. Tanto que tiene usos en campos tales como la medicina, la física, la química, la ingeniera, etc. Siendo su impacto en la sociedad enorme, y es por ello que el estudio de la materia a nivel atómico es necesario para el progreso en los campos mencionados anteriormente.

Uno de los grandes avances en este campo fue la invención del microscopio de efecto túnel (STM) en el año 1981, que proporcionó una visualización sin precedentes de los átomos y sus enlaces, consiguiéndose años más tarde la manipulación de átomos sobre una superficie, haciendo posible el estudio de superficies metálicas, semiconductoras, y otras superficies más, en torno al 0,1 nm de resolución lateral y 0.01 nm de resolución de profundidad, es decir a una escala atómica.

El STM tal y como indica su nombre está basado sobre el efecto túnel, este concepto surge de la mecánica cuántica en la que una partícula viola los principios de la mecánica clásica penetrando una barrera de potencial o de impedancia mayor que la energía cinética de la propia partícula.

Para la generación de imágenes STM el departamento de Física de la Materia Condesada de la Universidad Autónoma de Madrid, ha desarrollado un modelo propio basado en el formalismo de no equilibrio de Green-Keldysh, para la simplificación de estas simulaciones derivándolas en simulaciones menos complejas. Pero el coste computacional para realizar estas simulaciones es muy elevado, por ello en este TFG nos hemos propuesto paralelizar y optimizar la simulación para la mejorar el tiempo de esta, y así poder en un futuro llegar a hacer simulaciones mayores.

Palabras clave:

Átomos, efecto túnel, formalismo de no equilibrio, microscopio de efecto túnel (STM),nanotecnología,paralelizar

Abstract

At present nanotechnology and its applications are increasingly present in our daily lives. Both that has applications in fields such as medicine, physics, chemistry, engineering, etc. Being huge impact on society, and that is why the study of matter at the atomic level is necessary for progress in the fields mentioned above.

One of the great advances in this field was the invention of the scanning tunneling microscope (STM) in 1981, which provided an unprecedented view of atoms and their bonds, achieving years later the manipulation of atoms on a surface, making possible the study of metal surfaces, semiconductor, and other more surfaces around 0.1 nm lateral resolution of 0.01 nm and depth resolution, i.e. the atomic scale.

The STM as its name suggests is based on quantum tunneling, this concept comes from quantum mechanics in which a particle violates the principles of classical mechanics penetrating a potential barrier or higher impedance than the kinetic energy of the particle.

For STM imaging the department Matter Physics Condensed of the Universidad Autónoma de Madrid, it has developed its own model based on the formalism of non-equilibrium Green-Keldysh, to simplify this by deriving simulations in less complex simulations. But the computational cost for these simulations is very high, so in this TFG we have proposed to parallelize and optimize the simulation in order to improve the computational simulation time, so we can get in the future to make further simulations.

Keywords:

Atoms, quantum tunneling, formalism of non-equilibrium, the scanning tunneling microscope (STM), nanotechnology, parallelize

Índice

Introducción 1

2. Estado del Arte 4

2.1. Microscopio de Efecto Túnel 4

2.2. Enfoque del proyecto 4

2.3. Otros enfoques existentes 5

3. Análisis de Requisitos 6

3.1. Requisitos no funcionales 6

3.2. Requisitos funcionales 7

4. Metodología 8

4.1. Lenguajes de programación 8

4.2. Compiladores 8

4.3. Modelos de programación 9

4.4. Herramientas usadas 10

5. Análisis y Diseño del Proyecto 13

5.1. Análisis 13

5.2. Diseño 14

5.3. Paralelismo 17

6. Pruebas y Rendimiento 21

6.1. Primera parte 21

6.2. Segunda parte 28

6.3. Otras pruebas realizadas 33

7. Conclusiones 36

Anexo A: Fichero de salida Stm.out 38

Anexo B: Tablas de la Parte 1 42

Anexo C: Tablas de la Parte 2 44

Anexo D: Tablas de otras pruebas 46

Anexo E: Tablas de conflictos 47

Referencias 48

Índice de figuras

Ilustración 1-1 Imagen STM 1
Ilustración 1-2 Propagación electrónica en la muestra y/o punta 2
Ilustración 5-1 Diagrama de ejecución del programa paralelizado. 19
Ilustración 6-1 Gráfica rendimientos, parte 1 21
Ilustración 6-2 Gráfica de aceleración respecto al tamaño del problema, parte 1 23
Ilustración 6-3 Gráfica aceleración, parte 1 23
Ilustración 6-4 Gráfica eficiencia, parte 1 25
Ilustración 6-5 Gráfica costes, parte 1 26
Ilustración 6-6 Gráfica overhead en relación al tamaño del problema, parte 1..... 27
Ilustración 6-7 Gráfica overhead, parte 1 27
Ilustración 2-8 Gráfica rendimientos, parte 2 28
Ilustración 6-9 Gráfica de aceleración respecto al tamaño del problema, parte 2 29
Ilustración 6-10 Gráfica aceleración, parte 2 30
Ilustración 6-11 Gráfica eficiencia, parte 2 30
Ilustración 6-12 Gráfica costes, parte 2 31
Ilustración 6-13 Gráfica overhead en relación al tamaño del problema, parte 2..... 32
Ilustración 6-14 Gráfica overhead, parte 2 33
Ilustración 6-15 Gráfica de diferentes repartos..... 34
Ilustración 6-16 Gráfica diferencia de rendimientos 1 dimensión vs. 3 dimensiones . 34
Ilustración 6-17 Gráfica diferencia de rendimientos 1 dimensión vs. 3 dimensiones en el segundo clúster..... 35

Índice tablas

Tabla 4-1 Opciones del compilador Ifort *9*
Tabla 4-2 Afinidad entre cores *10*
Tabla 4-3 Opciones de la herramienta KMP_AFFINITY *11*
Tabla 5-1 Fichero de configuración *15*

Introducción

El Microscopio de Efecto Túnel (STM) [1,2]ya es toda una herramienta fundamental en física de la materia condensada. El STM consiste en producir una corriente túnel [3] entre una punta y una muestra aplicando una diferencia de potencial entre ambas. Desde aparición del STM en 1981 ha revolucionado el campo de la materia condensada, permitiendo explorar con una resolución atómica fenómenos como la topografía superficial, las propiedades electrónicas y vibraciones, así como manipular la materia a escala atómica.

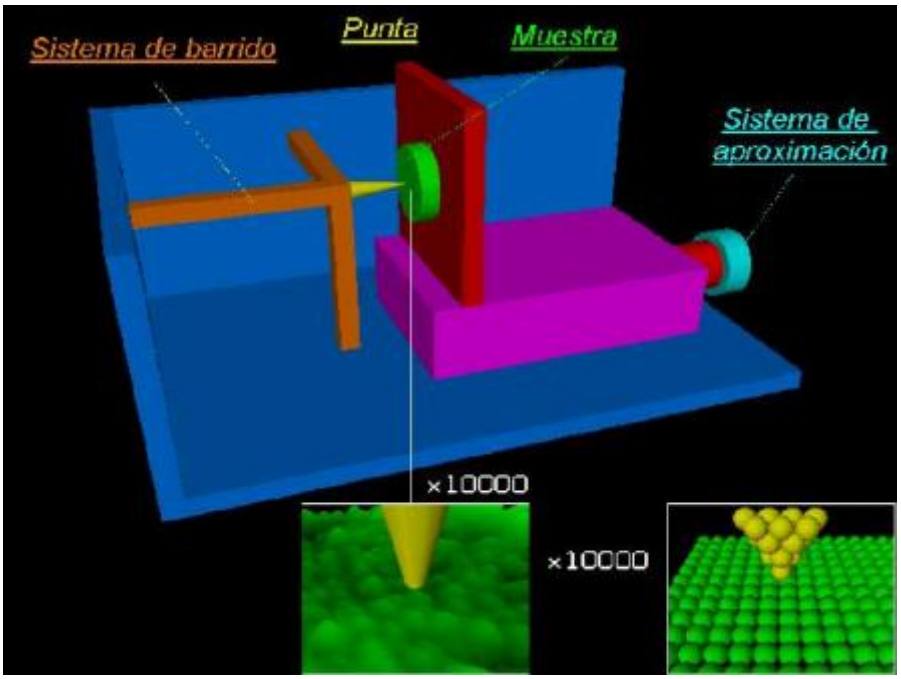

Ilustración 1-1 Imagen STM

La formación de imágenes en un STM está controlada por la corriente túnel y el voltaje aplicado entre la punta y la muestra. La realización de simulaciones computacionales de imágenes STM realistas requiere calcular esta corriente con precisión teniendo en cuenta la estructura atómica tanto de la punta como de la muestra. Para realizar este cálculo, uno de los métodos más precisos se basa en el uso de funciones de Green [2, 4] fuera del equilibrio usando el formalismo de Keldysh [2 , 4] . Este formalismo permite separar el problema general en un sistema de punta aislada, muestra aislada y acoplo punta muestra. Permite así incluir, de forma sencilla, cálculos precisos de la estructura electrónica de los electrodos. Además, este formalismo resuelve el problema del no equilibrio, de una manera efectiva, más allá de una teoría de perturbaciones mediante las funciones de Green. Dicho método incluye el múltiple scattering [2] a todo orden y todos los efectos inelásticos, utilizando una formulación muy

compacta para el cálculo de la corriente. Hoy en día se puede considerar este formalismo como el más preciso.

Con el formalismo de Green-Keldysh el acoplo punta-muestra se introduce a través de unos hoppings [2] efectivos que se definen como una suma de la inyección directa más los múltiples procesos de reflexión y propagaciones electrónicas en la muestra y la punta(Ilustración 1-2)

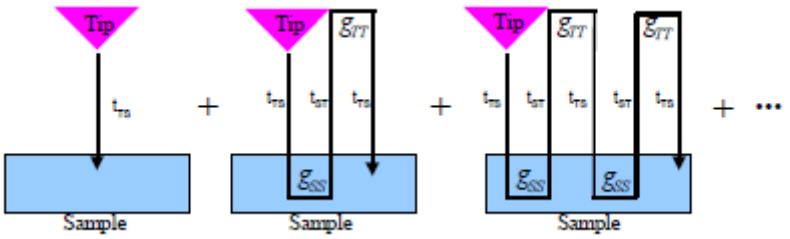

Ilustración 1-2 Propagación electrónica en la muestra y/o punta

Motivación del proyecto

El proceso del cálculo de la función de Green junto al cálculo de las corrientes túnel computacionalmente hablando tiene un costo muy grande, ya que imágenes STM de sistemas de tamaños pequeños, es decir superficies que se pueden simular con celdas de pocos átomos (<10), se pueden simular corriendo el código en serie, en un rango de minutos-horas (<10). Sin embargo si se quieren simular sistemas más grandes, complejos y/o con mayor precisión, como los demandado actualmente por los estudios realizados en el campo, los tiempos de simulación de una sola imagen STM se van al orden de semanas corriendo en serie. Por tanto, es necesario un incremento en la eficiencia de la herramienta de simulación. Por ello este trabajo pretende paralelizar y optimizar estos cálculos para una plataforma multicore, mejorando así el rendimiento de la plataforma STM.

Objetivos y Planificación

Este proyecto se enmarca en un contexto donde se desarrolla una investigación por parte del ScanningProbe-MicroscopyTheory&NanomechanicsGroup del departamento de Física Teórica de la Materia Condensada. Por ello, tras una serie de reuniones los objetivos de este trabajo son:

- Particularizar el tipo de paralelización del código.
- Paralelizar el cálculo de las funciones de Green

- Paralelizar el cálculo de la corriente túnel en función de la posición de la punta sobre la muestra en un mallado del espacio de donde se pueden extraer las imágenes STM.

Estructura del documento

El documento se divide en siete capítulos. El primer capítulo comprende una breve introducción al contexto y plantea los objetivos. En el segundo capítulo se sitúa al lector en el contexto de la aplicación. El tercer capítulo enumera y explica al lector los requisitos de sistema. En el cuarto capítulo se describe la metodología usada en el proyecto. El quinto capítulo muestra el análisis y diseño propuestos para el proyecto. En el sexto capítulo se muestran las pruebas y el análisis de rendimientos hechos en el proyecto y por último en el séptimo capítulo se exponen las conclusiones sacadas del proyecto así como los posibles desarrollos que se podrían realizar en el futuro.

2. Estado del Arte

En este apartado se comenta de forma general que es el Microscopio de Efecto Túnel (STM) y se explican distintos usos del STM en la actualidad.

2.1. Microscopio de Efecto Túnel

El Microscopio de efecto túnel (STM) es una técnica esencial de la nano-ciencia para obtener imágenes de resolución atómica de especies adsorbidas. STM ha encontrado aplicaciones en la física, la química y la biología, y es una de las pocas técnicas no destructivas que pueden tomar imágenes en el crecimiento in situ y procesos de adsorción con una resolución atómica. Incluso puede sondear estructuras electromagnéticas, estados de excitación colectiva, y los patrones de interferencia causada por las impurezas.

2.2. Enfoque del proyecto

Nuestro enfoque está basado en el uso del STM para la generación de imágenes, pero desarrollando un modelo propio basado en el formalismo de no equilibrio de Green-Keldysh y de la teoría funcional de la densidad(DFT) [2].

- **Formalismo de no equilibrio:** Este formalismo implementa, en Mecánica Cuántica, la evaluación de promedios y funciones de correlación en términos de un formalismo de funciones de Green. Siendo un promedio un caso especial de la función de correlación. Este formalismo permite una evaluación sistemática de las funciones de correlación, las cuales se relacionan de manera simple a los promedios de interés de un sistema dado.

- **Teoría funcional de la densidad:** El DFT se basa en la premisa de que la energía de una molécula puede determinarse a partir de la densidad electrónica en lugar de usar la función de onda.

  Historia:

  En 1927, Thomas y Fermi, Calcularon la energía de un átomo, representando su energía cinética como un funcional de su densidad electrónica, y combinando esto con las expresiones clásicas de las interacciones núcleo-electrón y electrón-electrón

2.3. Otros enfoques existentes

STM no se utiliza estrictamente para realizar una observación pasiva de especies de superficies, sino también para manipular los átomos y moléculas individuales de una manera ascendente, para construir sistemas a nano-escala. También puede ayudar a provocar complejas reacciones químicas en cadena de diferentes especies a la vez, seguidos por pulsos en lugares específicos. Siendo crucial la posibilidad de que el microscopio distinga entre los diferentes productos de la reacción química.

Usando este enfoque se han realizado múltiples experimentos, uno de ellos se basa en modificar directamente la composición química o la estructura de la superficie molecular, a la vez que sigue la influencia de la imagen resultante del STM sin realizar cálculos exigentes de la estructura electrónica en todo el sistema [5]. Donde un usuario externo realiza una modificación química o física en el modelo original del STM, facilitando la determinación de la estructura del absorbato, la composición química, la reconstrucción de la superficie, constitución del ápice, el ángulo de la punta, etc.

La principal característica de este enfoque es que es una combinación eficiente de la estructura electrónica y métodos electrónicos de transporte para producir rápidamente imágenes STM precisas.

3. Análisis de Requisitos

En este capítulo se expondrán los requisitos funcionales y no funcionales obtenidos tras las reuniones con el departamento de física teórica de la materia condensada.

3.1. Requisitos no funcionales

Compilador

Por dependencias de librerías de Intel, el programa ha de ser compilado con Ifor.

Rendimiento

El rendimiento del programa paralelizado ha de ser superior al rendimiento del programa en serie.

Tolerancia a fallos

El programa paralelizado ha de obtener resultados semejantes al programa en serie. Semejantes ya que se hace uso de operaciones en coma flotante, haciendo que la variación de resultados de uno a otro sea ínfima pero existente.

Escalabilidad

La paralelización ha de ser lo mayor escalar posible.

Usabilidad

El usuario debe ser capaz de ejecutar el programa en paralelo al igual que lo hacía en serie, la única diferencia de ejecución con la del programa en serie queda detallada en un posible manual de uso del programa, si es que hubiese alguna.

Interfaces

- Hardware: El sistema se debe ejecutar sobre una infraestructura con un sistema multiprocesador con una arquitectura que permita el uso de memoria compartida. También esta infraestructura debe contar con el compilador de Intel Ifor.
- Software:
    - Se requiere licencia de Intel para el compilador Ifor
    - El programa deberá funcionar sobre Fortran 90.
    - La paralización ha de ser sobre OpenMP.
    - Se requieren las librerías MPK de Intel y la librería OpenMP para Intel.

3.2. Requisitos funcionales

Estrategia de paralelización

El usuario no deberá de ser el que indique la estrategia de paralelizado, de ello se encargara el sistema.

Repartición del trabajo entre hilos

Independientemente del tamaño del trabajo a computar, el sistema y no el usuario se tiene que encargar de la distribución del trabajo entre hilos con la menor perdida de rendimiento posible.

4. Metodología

4.1. Lenguajes de programación

El Lenguaje de programación usado en el proyecto es Fortran [6], este es un lenguaje de alto nivel desarrollado por IBM en 1957. Este lenguaje fue creado con el objetivo de hacer más fácil el cálculo numérico y la computación científica, es decir una forma de hacer más fácil la traducción de fórmulas matemáticas a código. De ahí su nombre, Fortran acrónimo de "Formula Translation".

Desde la creación de este lenguaje se han publicado diversas versiones, en este proyecto se hará uso de la versión Fortran 90. La cual destaca por la posibilidad de operar con vectores, ya que en las versiones anteriores no se podía hacer uso de este tipo de operaciones, siendo estas la base de este proyecto.

4.2. Compiladores

Los Compiladores cuentan con herramientas de optimización que sirven para minimizar el tiempo de ejecución del programa. En un principio, el compilador que iba a ser utilizado para Fortran 90 iba a ser el proporcionado por GNU (gfortran) , pero por temas de librerías la compilación no era posible. Por lo que el compilador usado ha sido el proporcionado por la compañía Intel(R)(Ifort).

Ifort

Este compilador de Intel para Fortran incluye optimizaciones independientemente del procesador. No obstante, Ifort dispone de optimizaciones extra para los propios procesadores de Intel. Al igual pasa con las opciones del compilador Ifort, hay ciertas opciones reservadas para los procesadores de Intel.

En el clúster usado se cuenta con la versión 12 instalada y para esta versión las flags disponibles son las siguientes:

Flags	Descripción
-O0	Desactiva cualquier optimización.
-O1	Optimiza para obtener máxima velocidad desactivando aquellas optimizaciones que proporcionan un beneficio en la velocidad pequeño a costa de aumentar el tamaño del ejecutable.
-O2/-O	Optimiza para conseguir la máxima velocidad del ejecutable (valor por defecto).
-O3	Optimiza para conseguir la máxima velocidad del ejecutable incluyendo optimizaciones más agresivas que -O2. Estas optimizaciones pueden no mejorar el comportamiento de algunos programas.
-fast	Activa los siguientes flags: Las optimizaciones de -O3. El flag -xhost que utilizará el conjunto de instrucciones másavanzado para el procesador en el que se ha compilado. -ipo, añade optimización interprocedimental. • -no-prec-div -static.
-Ofast	Activa optimizaciones -O3 y -no-prec-div.

Tabla 4-1 Opciones del compilador Ifort

4.3. Modelos de programación

OpenMp

OpenMP [7] es una interfaz de programación de aplicaciones portable y escalable, la cual proporciona una interfaz para la programación multiproceso en entornos de memoria compartida.

Esta API está disponible para muchas arquitecturas y para distintas plataformas, desde un ordenador de escritorio hasta supercomputadoras. Lo bueno de esta API es que permite añadir concurrencia a los programas sobre la base del modelo de ejecución fork-join [8] de una forma fácil mediante directivas de compilador, rutinas de biblioteca y variables de entono.

Por todo ello se ha escogido esta API, por su sencillez, escalabilidad y uso de memoria compartida frente a MPI, que necesita interactuar mediante pasos de mensajes entre procesos.

Otra opción podría haber sido CUDA, ya que también hace uso de memoria compartida. Pero la programación en CUDA necesita de hardware especifico y su flexibilidad es menor.

4.4. Herramientas usadas

Intel Mkl

Intel MathKernel Library es una librería matemática altamente optimizada y *multithread* de Intel pensada para aplicaciones que requieren el máximo rendimiento. En nuestro caso usamos BLAS/LAPACK para optimizar la multiplicación de matrices y la resolución de un problema de auto-valores (diagonalización de una matriz).

kmp_affinity

Generalmente los trabajos se ejecutan dentro de un cpuset, teniendo un conjunto de hilos asignados a unos cores determinados, a pesar ello esta asignación dentro del cpuset puede variar, reduciendo así el rendimiento. Por ello se hace uso de esta herramienta proporcionada por Intel [9], la cual sirve para establecer la afinidad, ayudando así a simplificar la tarea.

Esta herramienta ha sido usada principalmente para repartir los hilos a unos cores específicos. Esto se ha hecho porque en el clúster utilizado, por características de hardware, había conflictos entre determinados cores. Estos cores competían por el acceso a memoria, y al hacer uso de esta herramienta se ha llegado a disminuir la penalización hasta en un 73% para 2 hilos. A continuación se muestra una tabla de la afinidad entre cores del clúster utilizado.

	0	1	2	3	4	5	6	7
0	X	259m4.249s	168m52.191s	168m47.327s	291m9.357s	259m7.413s	171m2.938s	170m54.605s
1		X	168m54.122s	168m53.005s	258m24.920s	291m8.858s	169m33.166s	169m46.166s
2		---	X	259m1.679s	170m34.849s	169m30.201s	291m31.740s	258m44.779s
3		---	---	X	169m7.297s	169m23.584s	259m29.079s	291m3.073s
4		---	---	---	X	259m7.957s	168m45.166s	168m48.006s
5		---	---	---	---	X	168m54.452s	168m53.100s
6		---	---	---	---	---	X	259m28.521s
7		---	---	---	---	---	---	X

Tabla 4-2 Afinidad entre cores

Para hacer uso de la variable de entorno KMP_AFFINITY hay que hacer uso de la siguiente sintaxis: KMP_AFFINITY= [<*modifier*>,...] <*type*> [,<*permute*>] [,<*offset*>](Tabla con los posibles valores para los diferentes argumentos se encuentra en la Tabla 4-3).

Argument	Default	Description
modifier	noverbose respect granularity=core	Optional. String consisting of keyword and specifier. • granularity=<specifier> takes the following specifiers: fine, thread, and core • norespect • noverbose • nowarnings • proclist={<proc-list>} • respect • verbose • warnings The syntax for <proc-list> is explained in mid-level affinity interface.
type	none	Required string. Indicates the thread affinity to use. • compact • disabled • explicit • none • scatter • logical (deprecated; instead use compact, but omit any permute value) • physical (deprecated; instead use scatter, possibly with an offset value) The logical and physical types are deprecated but supported for backward compatibility.
permute	0	Optional. Positive integer value. Not valid with type values of explicit, none, or disabled.
offset	0	Optional. Positive integer value. Not valid with type values of explicit, none, or disabled.

Tabla 4-3 Opciones de la herramienta KMP_AFFINITY

Para asignar los hilos a los cores hay que elegir la opción "proclist" y asignar el número del core en la posición del vector que queramos, siendo este el número del hilo:

$$\text{KMP\_AFFINITY} = \text{"explicit, proclist} = \ [0,2], \text{verbose"}$$

En este caso estamos asignando al primer hilo el primer core de la cpu y el tercer core al segundo hilo. También se puede que elija entre varios:

$$\text{KMP\_AFFINITY} = \text{"explicit, proclist} = \ [0,2-4], \text{verbose"}$$

En este otro caso el segundo hilo elegiría desde el tercer core al quinto.

Numdiff

Herramienta de software libre la cual puede ser redistribuida y/o modificada bajo los términos de GNU General PublicLisense .Esta herramienta compara archivos línea por línea, haciendo caso omiso de las pequeñas diferencias numéricas y/o de diferentes formatos numéricos indicados por parámetros. Al operar con operaciones en coma flotante el resultado

en las versiones paralelas siempre va a diferir de la versión en serie, por eso se usa esta herramienta, para diferenciar con un rango establecido los resultados obtenidos.

Clúster

Herramienta proporcionada por el departamento de física teórica de la materia condensada en el cual se ha realizado el proyecto y trata de conglomerado de 7 nodos con 8 cores Intel Xenon x5450 a 3.00 GHz cada uno y 32 Gb de memoria RAM por nodo, es decir 4 Gb/core. En este clúster se dispone de la versión del compilador Ifort 12.0.

En los últimos días se ha accedido a un nuevo clúster para poder hacer algunas pruebas más sobre escalabilidad, ya que este nuevo clúster tiene hasta 12 cores por nodo. Estos 12 cores están separados en dos lotes de 6, siendo cada core un Intel(R) Xenon(R) E5649 a 2,53 GHz con 12 Mb de cache y con 4 Gb/core de RAM.

5. Análisis y Diseño del Proyecto

Este capítulo describe el análisis y diseño desarrollado durante este trabajo. El objetivo es realizar una paralelización óptima del programa, para ello se ha hecho un análisis exhaustivo del código en el cual se ha evaluado si el código es paralelizable, así como las dependencias dentro de este y la mejor estrategia de paralelización posible.

5.1. Análisis

En primer lugar, para paralelizar un código usando memoria compartida primero se ha de identificar las zonas a paralelizar, una vez encontradas hay que encontrar todas las dependencias de tipo RAW - ReadAfterWrite [10], RAR - ReadAfterRead [10] y WAR - WriteAfterRead [10] existentes.

Para poder encontrar las dependencias he tenido que familiarizarme con Fortran y con el uso de OpenMp en él. Teniendo que familiarizarme con el uso, la manipulación de subrutinas, aprendiendo como es su declaración y el paso de variables como argumento en ellas, también familiarizarme con el uso de memoria dinámica en este lenguaje, la manipulación de vectores, y también con el uso y manipulación de módulos. Todo ello me ha servido para la comprensión del código, ayudándome a encontrar y resolver cada una de las dependencias existentes en él.

Una vez encontradas y solventadas todas las dependencias se realiza un estudio de rendimientos con diferentes estrategias (reparto estático, dinámico, con diferentes tamaño de reparto, etc.) para diferentes tamaños del problema. Se harán tests de rendimientos para cada estas estrategias, en la que los resultados se mostraran y discutirán en un capítulo dedicado a ello.

Cabe mencionar que al dividirse la paralelización en dos partes independientes entre sí, se ha hecho lo mismo con análisis. Por ello se ha realizado un estudio de dependencias y estrategias independiente para cada una de las partes resolviéndose así por separado.

5.2. Diseño

Dentro de este apartado, se expondrá la estructura del programa, de los ficheros de entrada y los ficheros de salida generados en las zonas paralelizadas del programa los cuales serán estudiados para verificar que la paralelización es correcta, así como la estrategia llevada.

Estructura del programa

La primera parte del programa trata de la lectura del fichero de configuración, stm.inp, este fichero nos proporcionará la información sobre el tipo y el tamaño del cálculo.

Una vez el programa tiene los parámetros de entrada el siguiente paso es la lectura de toda la información acerca de la muestra y de la información necesaria para la resolución de los cálculos pertinentes(lectura de la estructura geométrica, información Hamiltoniana y de la superposición, tamaño de los puntos K).

Con esa información el programa realiza el cálculo de las coordenadas de las diferentes celdas y calcula para la muestra la Función de Green y de DOS(estos cálculos son los que posteriormente paralelizaremos).

Ahora que el programa ha realizado los cálculos para la muestra, este realiza los mismo pasos para calcular la punta. Realizando una lectura de la información sobre ella y sobre la información necesaria para la resolución de los cálculos. Una vez obtenidos los datos el programa calcula la Función de Green y de DOS, siendo esta vez para la punta(estos cálculos por su complejidad también serán posteriormente paralelizados).

Una vez con los cálculos de la muestra y de la punta el siguiente paso es escribir dichos resultados en los ficheros de salida pertinentes(GreenFunc_smp.out y GreenFunc_tip.out), los cuales son analizados posteriormente como ya se ha mencionado anteriormente.

Esto sería básicamente la primera parte del programa, la segunda parte trata sobre el cálculo de las corrientes según el formalismo de Keldysh (estos cálculos están sometidos a paralelización), para realizar este cálculo el programa hace una interpolación partiendo de los valores de la función de Green , de DOS y de la lectura de los hopping.

Una vez calculados son escritos en los ficheros current_sample_nden.out y current_sample_deno.out dependiendo de si en el fichero de configuración se ha elegido la opción de denominador, de no denominador, o de ambos.

Ficheros de entrada

Solo hay un fichero de entrada, el fichero de configuración este fichero está diseñado para establecer los parámetros del programa que a su vez determina el tamaño del problema a calcular, tal y como se ha mencionado en el apartado anterior.

Un ejemplo de la estructura del fichero es la siguiente:

0.00	**1.00**	**64**	**0.05**
-1.0	**0.00**	**64**	**0.05**
3			
0	**0**		
0			
1			
0			
2			
0			
0			
100000000.d0			
1.86254800	**0.00000000**	**6**	
1.31692300	**0.00000000**	**6**	
9.31144800	**9.31144800**	**1**	
1	**1**		

Tabla 5-1 Fichero de configuración

Para la primera fila tenemos los valores para la einitial, efinal, nenergy, eimag para el sample, es decir para la muestra. Realmente con estos datos se nos dice que la muestra será una matriz de 64x 64 con un rango de valores de la energía de 0-1 para los valores reales y de 0.5 para los imaginarios. Para la segunda fila tenemos lo mismo pero para la punta.

En la tercera fila se puede poner un valor entre 1 y 3 dependiendo de este valor IREAD_ATOM hará una cosa u otra. Para el valor 1 seria la opción leer Atomo_i, para el valor 2 seria leer H_OLP de FB y en nuestro caso para el valor 3 leer H_OLP de OMX.

Para la cuarta fila hay dos opciones o 0 o 1, si es cero que es nuestro caso el programa calculara y escribirá en fichero la función de Green para la muestra o para la punta si fuese 1 el programa cargaría desde un fichero esta función. El número de la primera columna corresponde a la muestra y el de la segunda a la punta.

Para la quinta fila IDOS puede tomar 3 valores entre el 0 y el 3, siendo 0 la opción STM calculation, 1 la opción solo calculo de DOS y 2 el cálculo de las dos.

En la sexta fila se determina el modo de escaneo siendo 0 para los orbitales de la punta (tip orbitals), 1 para los orbitales de la muestra (simple orbitals) y 2 para ambos.

En la séptima fila se trata de IWRT_ORB esta variable puede valer o 0 o 1, si vale 1 el programa escribirá los valores de la órbita de la muestra en un fichero, en caso contrario, no.

En la octava fila elegiremos si los cálculos de las corrientes los haremos con denominador, sin denominador o con ambos, el cálculo con denominador es mucho más preciso, pero el coste de computación incrementa demasiado, por el lado contrario el cálculo sin denominador es menos preciso pero el coste computacional es muy pobre.

Para la novena fila se elige si se quiere calcular la curva STS siendo 0 para desactivar esta opción y 1 para activarla.

Seguimos con la décima fila, en esta fila se indica el formato de hopping siendo 0 el de la punta-muestra (tip-sample) y 1 el de la interacción fireball.

En la undécima fila se establece el valor de la energía de la ventana del STM.

En las 3 filas siguientes se establecen los valores mínimos y máximos de cada una de las dimensiones y el tamaño de estas, siendo para la fila duodécima el eje x, para la decimotercera fila el eje y, y para la decimocuarta fila el eje z.

Y por último en la decimoquinta fila se establecen el número de celdas a escribir.

Ficheros de salida

Los ficheros de salida que analizaremos son 4, los dos generados en los cálculos de la función de Green para la muestra y la punta, y los dos generados para el cálculo de las corrientes con o sin denominador.

La estructura de los ficheros de la función de Green es la siguiente:

```
+++ Energy #         1 :   5.58078248413778
+++ Cell:         1         1
+++ Atoms:         1         1
0.781596825634E-01    0.273311282053E-07  -0.504684172582E-03  -0.694947661133E-06  -
0.350283755969E-02   -0.909739001742E-07   -0.283452450308E-02       0.872225521987E-06
0.178470293938E-02 ...
... más valores.
```

+++ Atoms: 1 2

... Valores de Green para esta zona.

En la primera línea se indica el rango de la energía, a partir de ahí se indican la zona en la que estas y los datos de esta.

La estructura ara los ficheros generados en el cálculo de las corrientes es la siguiente.

- La cabecera:

```
WSxM file copyright Nanotec Electronica
WSxM ASCII XYZ file
X [A]  Y [A]  Z [A]
```

- Y los datos:

```
1.862548   1.316923   3.000000  0.11970E-02
1.862548   1.316923   3.500000  0.31148E-03
1.862548   2.822010   3.000000  0.99667E-03
```

Siendo la primera columna el valor de los ejes x, la segunda columna el valor de los ejes y, la tercera los valores de los ejes z y la cuarta columna los valores de los resultados de los cálculos de las corrientes.

Cabe mencionar que hay otro fichero generado de salida, el stm.out, este fichero nos muestra los pasos realizados por el programa principal, este fichero es solo de mera información y nos indica la ejecución del programa paso por paso. Este fichero no es relevante ya que los ficheros que contienen los datos importantes, los datos los cuales han sido el objetivo del programa, son los cuatro mencionados antes. No obstante, la estructura del fichero stm.out se expone un ejemplo en el anexo A.

Como se puede apreciar en este ejemplo del fichero stm.out se muestra cada una de las acciones del programa.

5.3. Paralelismo

Dentro del código hay dos zonas de alto coste computacional. Las cuales se han ejecutado de forma paralela para mejorar y reducir el tiempo de ejecución.

Previamente se ha hecho un análisis de estas zonas, el cual se ha comentado en el apartado anterior.

Tipo de paralelización

La arquitectura paralela utilizada se basa en el modelo MIMD [11](MultipleInstruction, Multiple Data) en la variante de memoria compartida, la cual fue definida por Flynn en una propuesta sobre clasificación de arquitecturas de computadores (Taxonomía de Flynn). Esta arquitectura tiene la ventaja del uso de memoria compartida, evitando el paso de datos entre procesadores y reduciendo el coste de ejecución.

Se propone una paralelización de grano fino, al ser la paralelización a nivel de bucles y sentencias, ya que la paralelización de grado fino ofrece una flexibilidad del tratamiento de datos. También al usar este modelo se permite que varios procesos se ejecuten en un espacio de direcciones común, es decir mayor grado de paralelismo, pero esto implica mayor gasto de entrada y almacenamiento de datos. Todo ello se manifiesta en un mayor número de variables y llamadas a subrutinas implicando una mayor penalización en la comunicación y en la planificación, reduciendo así la escalabilidad.

Implementación

Tras el estudio y solvencia de las dependencias encontradas con los cambios de código pertinentes, se han estudiado varias planificaciones de reparto de trabajo entre los hilos. Tras los estudios realizados del reparto de trabajo se ha optado por una planificación dinámica con reparto de tamaño uno, es decir a cada hilo se le asigna una iteración y en cuanto es acabado se le asigna una nueva.

Esta planificación nos proporciona un equilibrio de carga, por lo que ningún hilo se quedará en estado ocioso esperando a que los otros hilos terminen. El único impedimento es que este reparto requiere más trabajo, aunque el trabajo extra es despreciable en relación a la mejora conseguida.

Tras esta planificación el diagrama de ejecución del programa es el siguiente:

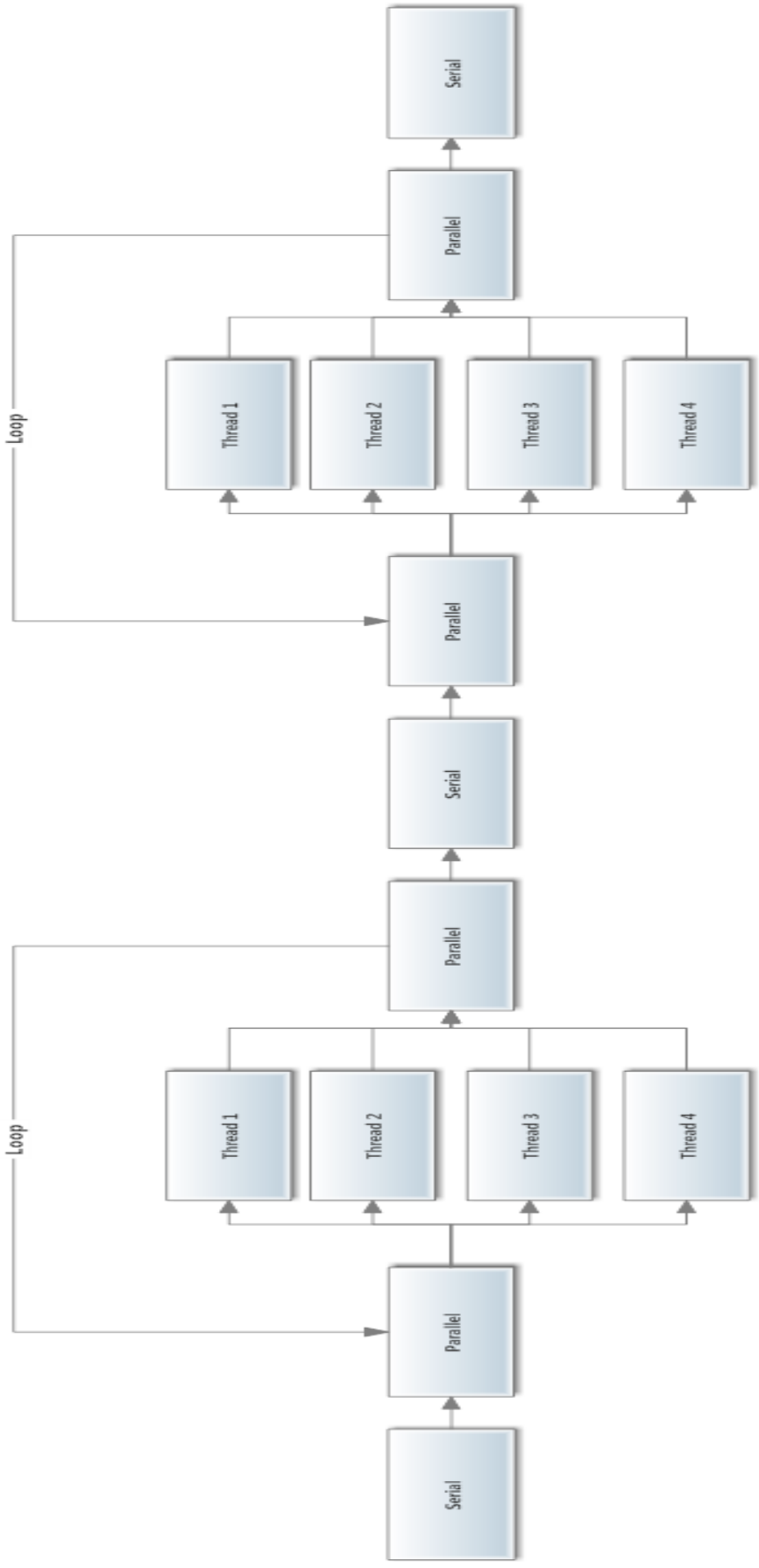

Ilustración 5-1 Diagrama de ejecución del programa paralelizado.

Como se puede apreciar en el diagrama las zonas paralelizadas son dos, tal y como se explico en el apartado del análisis.

En la primera parte se encontraron alrededor de 20 dependencias RAW las cuales fueron solventadas fácilmente haciéndolas privadas al hilo. Algunas, al estar instanciadas con memoria dinámica se han tenido que cambiar la zona de reserva y liberación de memoria, llevándolas a la zona paralelizada, para que se reservara y liberara en cada hilo.

En la segunda parte ha sido bastante más difícil el solventar las dependencias, ya que se hace uso de múltiples variables globales instanciadas en otros módulos y de varias subrutinas, todo ello a parte de las dependencias de las variables locales. Esto ha requerido múltiples modificaciones en el código.

Para resolver el problema de las variables globales se ha llevado a cabo dos estrategias dependiendo cada una de ella del coste computacional.

La primera estrategia se trata de cambiar las variables globales por locales y de pasar estas variables como argumentos a las subrutinas, esto requiere más uso de memoria ya que será proporcional al número de hilos, pero es una de las pocas soluciones viables ya que el coste computacional para el cálculo de estas variables es muy grande y no se pueden establecer en zonas criticas.

Por otro lado la segunda estrategia es la de calcular las variables globales en zonas criticas, esta estrategia se lleva a cabo cuando el coste computacional es insignificante.

Una posible alternativa a las estrategias antes mencionadas, es añadiéndole a todas estas variables globales una dimensión más en la que se le asignaría una dimensión a cada hilo, pero como estas variables son de varias dimensiones se ha optado por las estrategias anteriores.

También se ha tenido que modificar el código porque la zona a paralelizar se trataba de un triple bucle, esto hacía que la escalabilidad del problema dependiese del tamaño del bucle paralelizado, por lo que se optó a cambiar el triple bucle por un único bucle. Este cambio hace que el reparto de trabajo y el balanceo de carga sea más optimo, haciendo también más escalable el problema y aumentando así el rendimiento del programa. La única pega es que para el cálculo de los índices se realizan en una zona critica, pero como el cálculo de índices es ínfimo el cuello de botella generado por esta zona critica es insignificante.

6. Pruebas y Rendimiento

En este apartado se explican las pruebas realizadas al sistema para comprobar su correcto funcionamiento, así como el rendimiento de las diferentes estrategias de paralelización llevadas.

Tal y como se hizo el análisis separado en dos partes, las pruebas y los test de rendimientos también han sido separadas en dos partes.

Todas las tablas con los datos de las pruebas se hallan en el apartado de anexos.

6.1. Primera parte

El primer análisis de rendimientos trata del rendimiento de los tiempos de ejecución. Se han realizado para distintos tamaños de la muestra y de la punta, en el primer test los tamaños son de 64x8, en el segundo test los tamaños son de 64x16, en el tercer test de son de 64x32 y en el último test de 64x64.

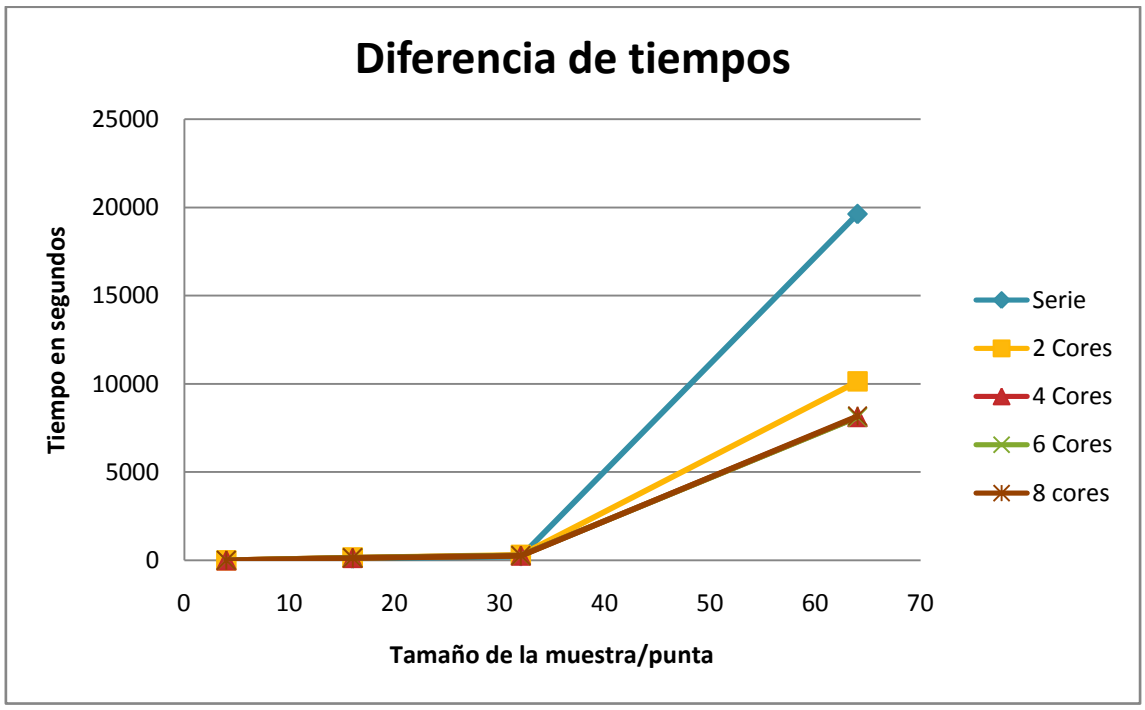

Ilustración 6-1 Gráfica rendimientos, parte 1

Como se puede apreciar en la gráfica a medida que va aumentando el trabajo a paralelizar se van separando los tiempos del programa en serie, por lo que a medida que el

trabajo aumenta la aceleración del programa aumenta hasta un tope de mejora. Esto viene enunciado en la ley de Amdahl [12].

También se puede observar una cosa muy importante y es que a partir de dos hilos la paralelización no mejora, esto se debe a que en el clúster en el que trabajamos tiene lo que se cree un problema de hardware, en el que determinados cores penalizan a otros a la hora de acceder a memoria.

También hay que decir que cuantos más cores a la hora de paralelizar el acceso y uso de memoria es proporcional al número de cores usados. Es decir si usamos dos cores, usamos el doble de memoria y de acceso a esta en las zonas paralelizadas. Si usamos cuatro cores estaríamos usando cuatro veces más uso y acceso a memoria, es por ello que a medida que aumentamos la paralelización el rango de mejora baja con respecto a lo esperado llegando a estancarse en un máximo.

Para poder medir la aceleración antes mencionada se debe usar la función de speed-up:

$$SpeedUp = \frac{Timepo\ Secuencial}{Tiempo\ Pparalelo}$$

Esta función nos muestra la mejora del programa paralelizado contra el serie, por lo que a mayor aceleración mayor rendimiento se obtendrá.

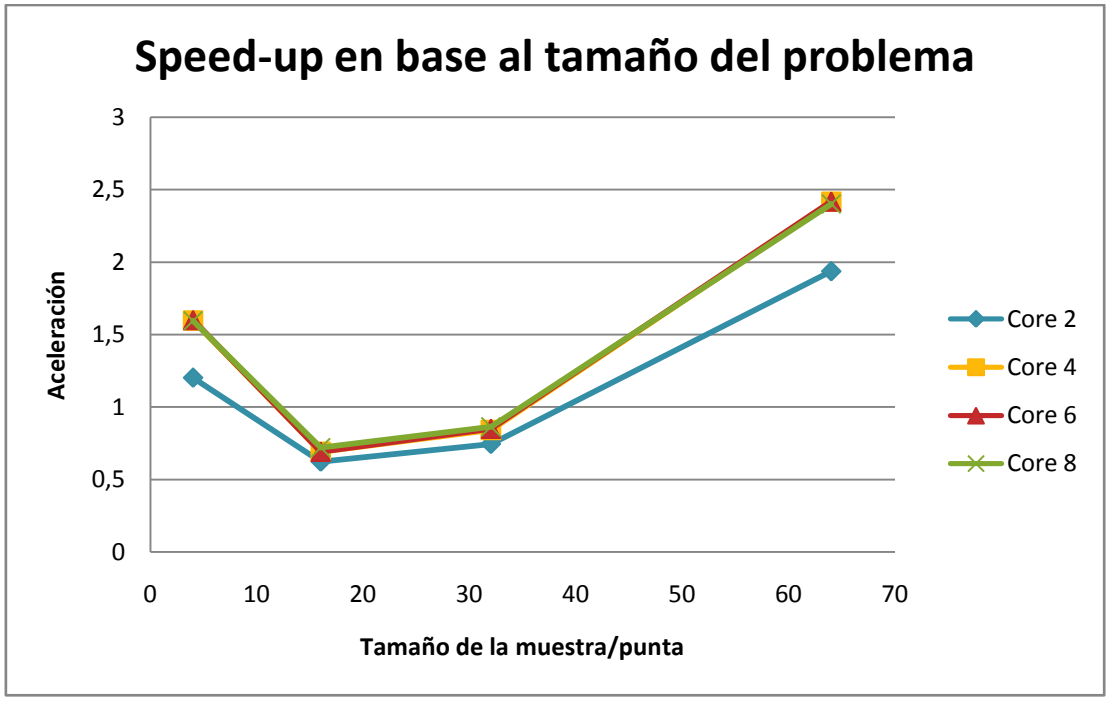

Ilustración 6-2 Gráfica de aceleración respecto al tamaño del problema, parte 1

Como se puede observar en esta gráfica el factor del trabajo es también muy importante ya que con poco trabajo el programa paralelizado no tiene una mejora apreciable, pudiendo llegar como es el caso a una disminución del rendimiento. Aunque con mayor trabajo se puede apreciar una mejora considerable, rozando lo ideal para dos hilos.

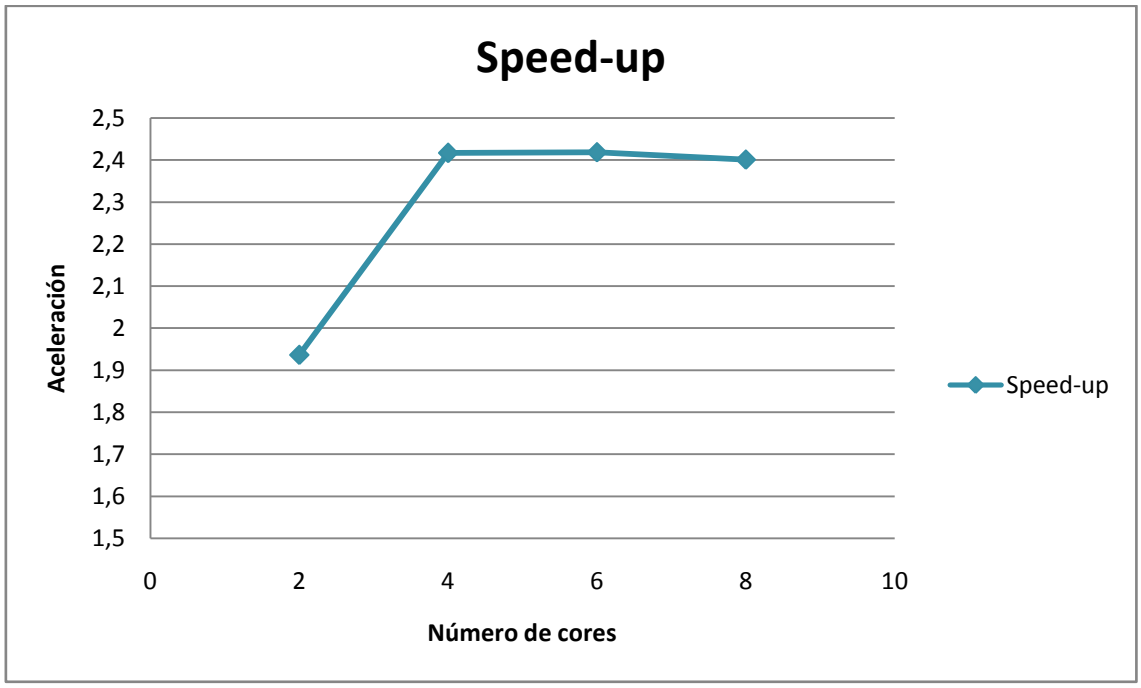

Ilustración 6-3 Gráfica aceleración, parte 1

Aquí se puede aprecia lo que antes comentábamos, y es que la paralelización es casi ideal con dos hilos consiguiendo casi un x2, pero ya a partir de dos hilos empieza a penalizar, consiguiendo prácticamente la máxima paralelización con cuatro hilos y a partir llega al tope.

Realmente, lo ideal para cuatro cores seria un speed-up de x4, pero eso en la realidad es imposible por lo antes comentado sobre el aumento de uso y acceso a memoria. Pero los resultados no deberían ser tan malos.

Ahora bien si nos fijamos, en la tabla del anexo E veremos que es imposible hacer una elección de cores en los que no haya una penalización grande y si a ello le sumamos el incremento de uso y acceso a memoria hacen que los resultados sean tan poco óptimos con respecto a los esperados. Y ya si nos vamos a seis cores veremos que es imposible el que no haya conflicto alguno entre ellos. Y es por ello que no haya mejora con respecto a cuatro cores, ya que también aumenta el uso y acceso a memoria con respecto a la paralelización con cuatro cores.

El siguiente análisis trata sobre la eficiencia de la paralelización, este análisis trata sobre una relación entre el speed-up, antes calculado, y el número de procesadores utilizados. Como se podrá observar la paralelización a partir de dos hilos no será eficiente por el problemas antes descrito. Su función es la siguiente:

$$Eficiencia = \frac{SpeedUp}{procesadores}$$

Se entiende que si el resultado de la división tiende a cero, significara que el paralelismo no es eficiente. Por el lado contrario, si la eficiencia tiende a uno la paralelización si es eficiente.

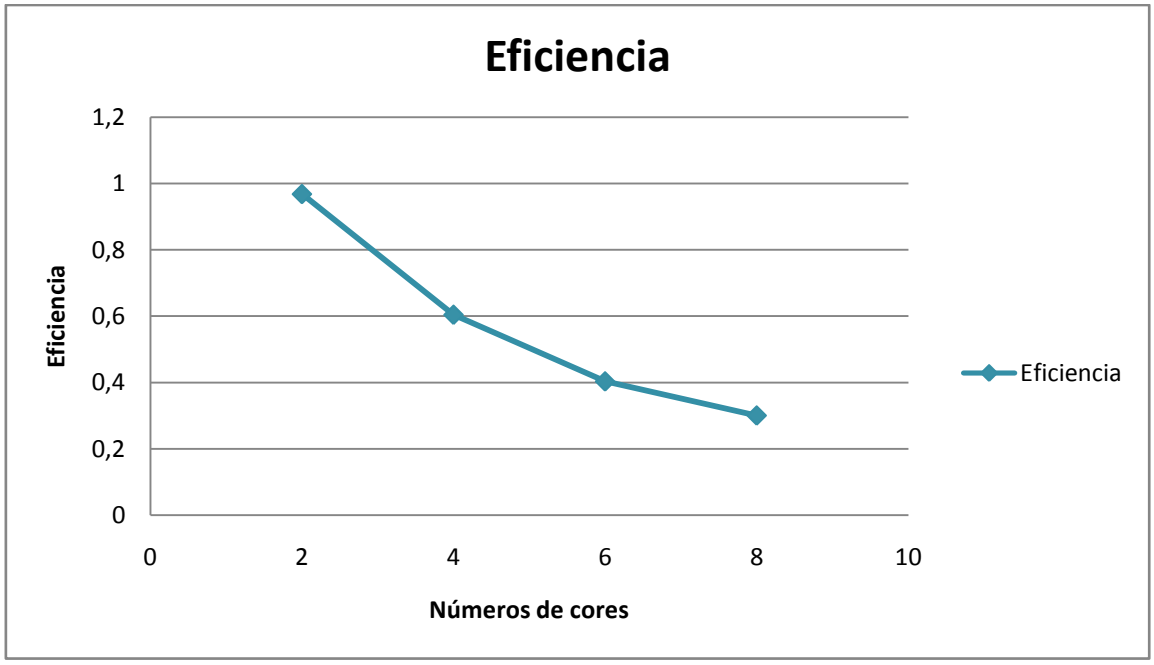

Ilustración 6-4 Gráfica eficiencia, parte 1

Como se ve en la gráfica pasa lo que comentábamos antes, con dos hilos si es eficiente, pero a partir de los dos cores la paralelización deja de serlo. Esto viene dado por el problema antes descrito, aunque aumentando de forma considerable el trabajo la eficiencia del programa puede llegar a mejorar.

Tras el análisis de la eficiencia ahora tratamos con el coste de la paralelización, calculándose de la siguiente manera.

$$Coste = procesadores * Tparalelo$$

Para que un paralelismo sea óptimo el coste del programa en paralelo debe ser igual al coste del programa en serie, en este caso por el problema del hardware, el coste a partir de dos hilos no es muy elevado por lo que la paralelización empieza a ser costosa, es decir deja de ser óptima.

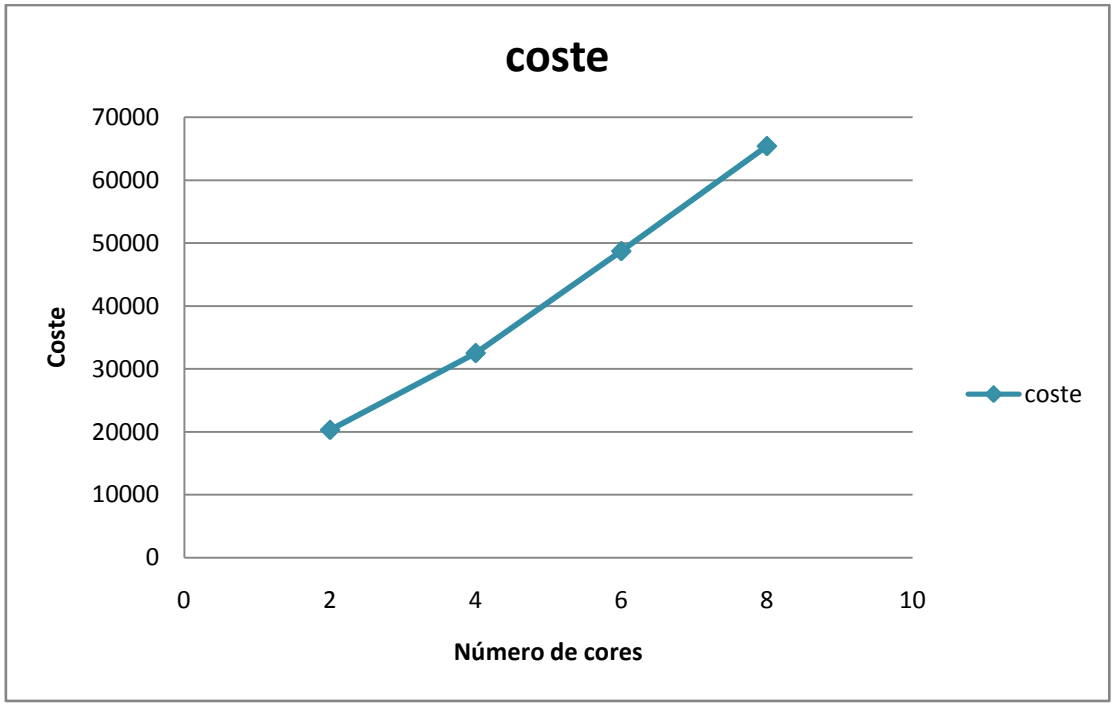

Ilustración 6-5 Gráfica costes, parte 1

Se puede observar que a medida que crece de forma lineal al número de cores, comprensible, ya que como se vio en la gráfica del speed-up, a partir de dos cores la aceleración incrementa muy poco.

Y por último tratamos con la medición del overhead. Este análisis mide la suma del tiempo de esperas, el exceso de computación, el exceso en las comunicaciones, etc. Producidas en la sincronización y accesos a memoria en la paralelización. Calculándose de la siguiente manera:

$$Overhead = procesadores * Tiempo\ Paralelo - Tiempo\ Secuencial$$

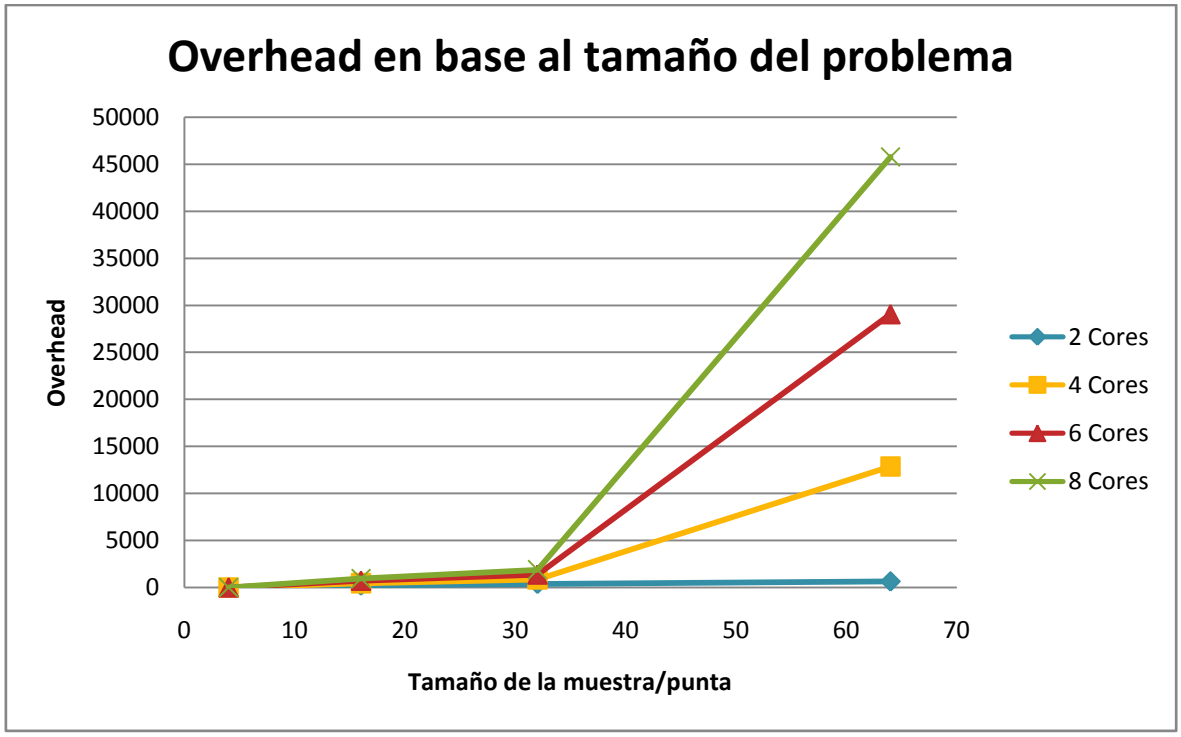

Ilustración 6-6 Gráfica overhead en relación al tamaño del problema, parte 1

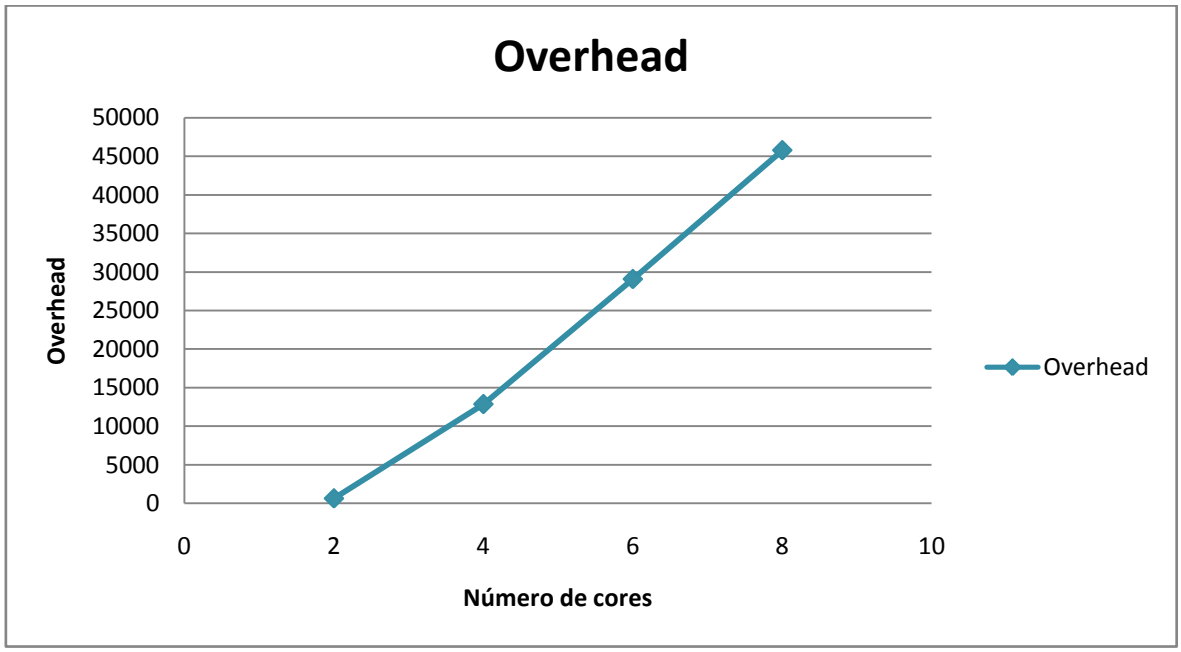

Ilustración 6-7 Gráfica overhead, parte 1

Como se observa en estas dos gráficas el coste con dos cores es prácticamente inexistente, pero a medida que el número de cores es mayor se corrobora que el coste generado por la penalización en el acceso a memoria entre cores conflictivos es muy elevado.

6.2. Segunda parte

En la segunda parte el análisis de rendimientos trata sobre el cálculo de las corrientes. Este cálculo se ha realizado para distintos tamaños, con el uso de denominador y de no denominador, ya que el coste de trabajo para no denominador es muy pobre el análisis ha sido realizado para el cálculo de ambos.

Los tamaños de las corrientes en sus tres dimensiones analizados son 4x4x1, 6x6x2 y 8x8x2.

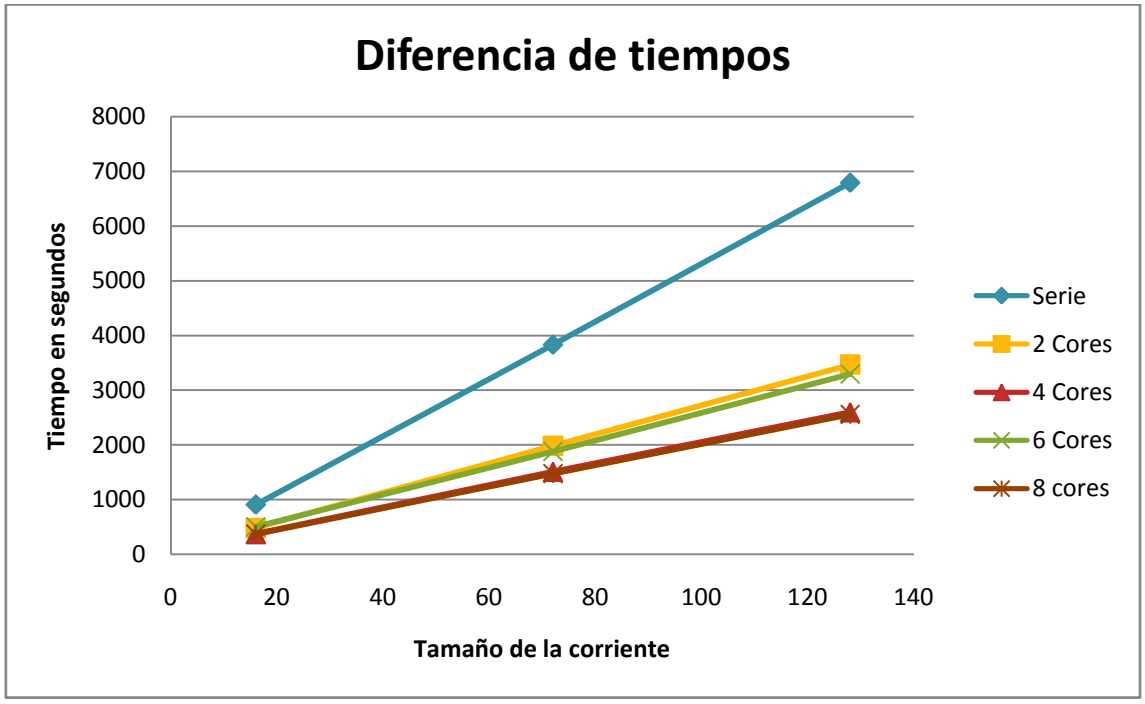

Ilustración 2-8 Gráfica rendimientos, parte 2

En la gráfica se observa que la carga de trabajo que requiere el cálculo de la corriente es mucho superior al cálculo de la función de Green, tanto que la paralelización a tamaños pequeños si es eficiente. A pesar de ello, los problemas de hardware persisten, esto se refleja en los tiempos con cuatro y ocho cores, los cuales son parejos y no se distancian mucho del tiempo con dos cores. Hay que decir que aunque estos problemas persisten, la mejora con cuatro cores es más eficiente que en la anterior parte. Esto se debe gracias a que la complejidad del problema es mayor, tal y como se mencionó antes. Y aunque se haga el doble de uso y acceso a memoria, se dobla la capacidad de computo, y como en esta parte la computación tiene un alto coste, hace que la penalización en el acceso y uso a memoria sea menor con respecto a la ganancia de computación, consiguiéndose así un mayor speed-up.

Un aspecto importante respecto al rendimiento con seis cores, es que al ser el tamaño de trabajo múltiplo de ocho la distribución con seis cores hace que sea muy poco eficiente. Se puede observar en la gráfica que es bastante menor el rendimiento que con cuatro y ocho cores, más adelante esto se verá reflejado en la gráfica de Speed-up. Esto se debe a la estrategia de reparto, ya que el total de lotes a repartir no es múltiplo de seis, por lo que algunos cores van a iterar más, dejando el reparto en no equitativo.

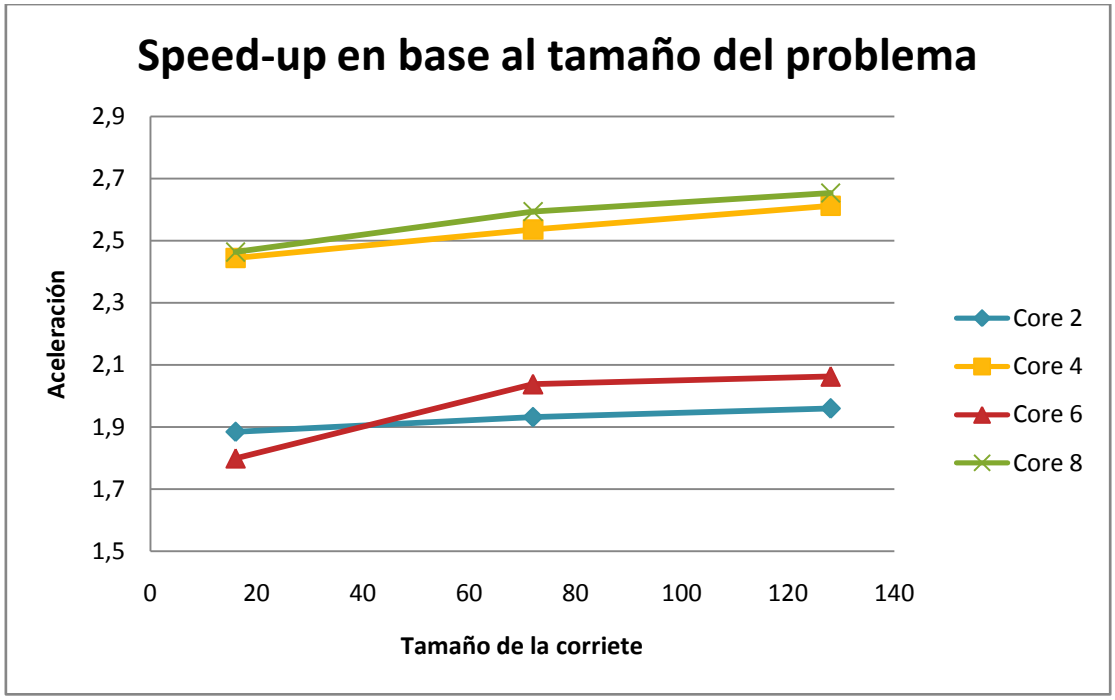

Ilustración 6-9 Gráfica de aceleración respecto al tamaño del problema, parte 2

Aquí se puede ver lo antes comentado sobre la paralelización con seis cores, llegando a ser menos eficiente que con dos cores a tamaños de trabajo pequeños, por lo demás la aceleración es bastante buena con dos cores y a partir de cuatro se colapsa por el problema de hardware.

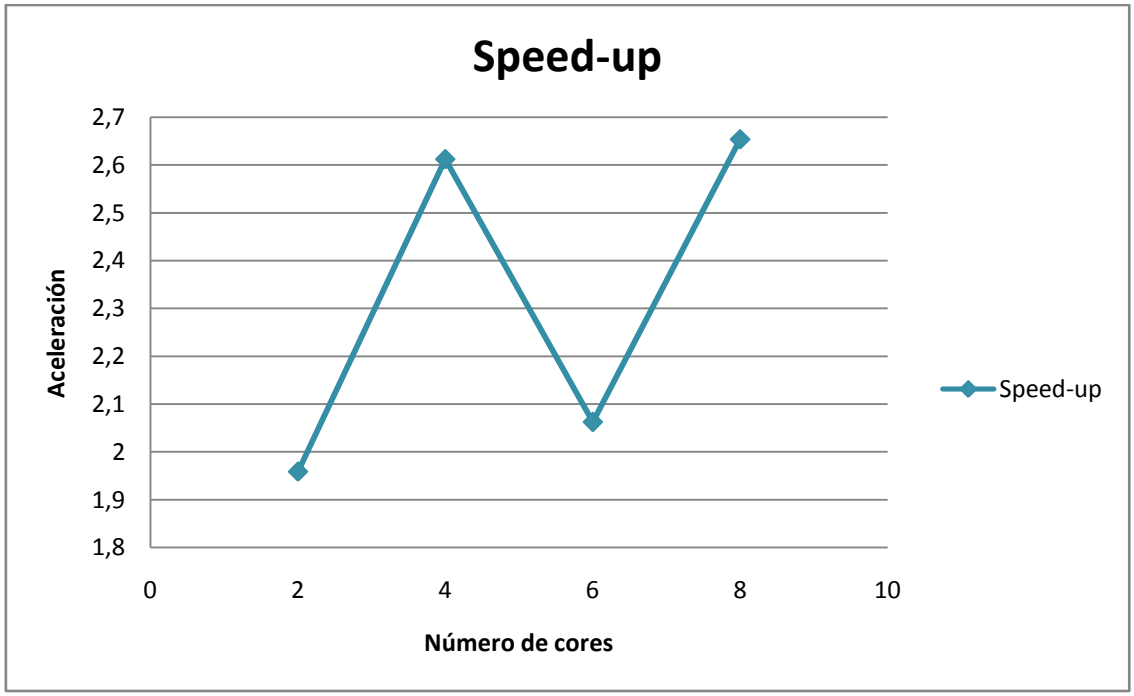

Ilustración 6-10 Gráfica aceleración, parte 2

En la gráfica de speed-up sobre un tamaño fijo se notablemente todo lo comentado anteriormente sobre la paralelización con seis cores y como se colapsa al llegar a cuatro. Pero, aun así el esta parte consigue una mayor aceleración que la anterior.

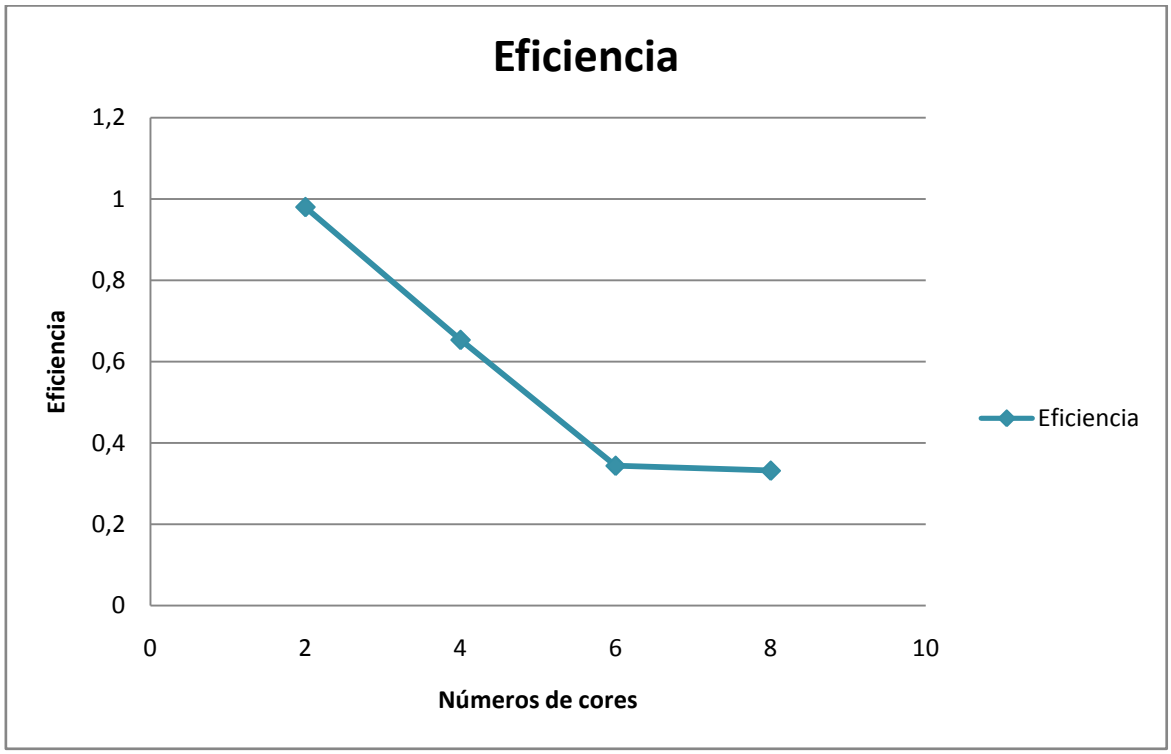

Ilustración 6-11 Gráfica eficiencia, parte 2

En la gráfica sobre la eficiencia de la paralelización de la segunda parte se puede observar como con dos cores es completamente eficiente y a partir de ahí baja, tal y como pasaba en la primera parte. Aunque con cuatro y ocho cores la eficiencia en esta parte es mayor que en la primera parte, ya que como se comento antes la penalización de hardware es menor por la gran cantidad de computo.

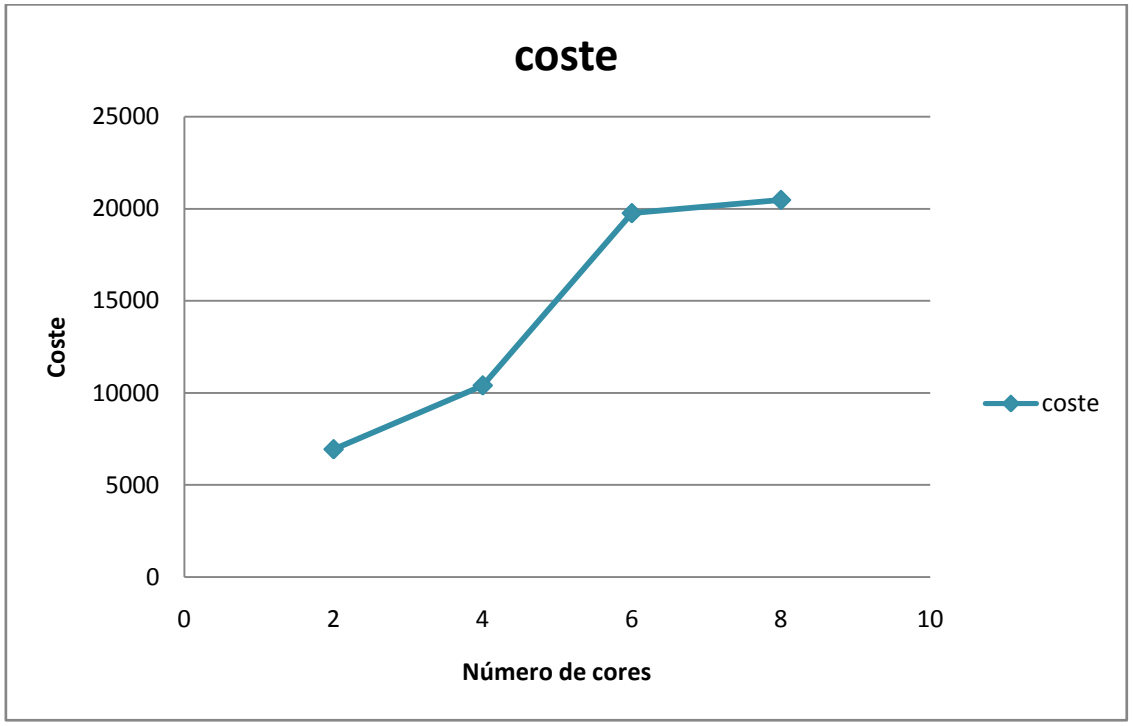

Ilustración 6-12 Gráfica costes, parte 2

Como se puede apreciar en esta gráfica el coste no es tan lineal como en la primera parte, esto se debe al problema con la distribución con seis cores el cual hace un salto gigantesco eliminando la linealidad del crecimiento. También hay que decir si comparamos la dos gráficas de coste es que el coste es bastante menor con relación a la primera parte, esto es debido a que la aceleración del programa, salvo con seis cores ,es mejor.

En las siguientes gráficas veremos el comportamiento del overhead en esta parte.

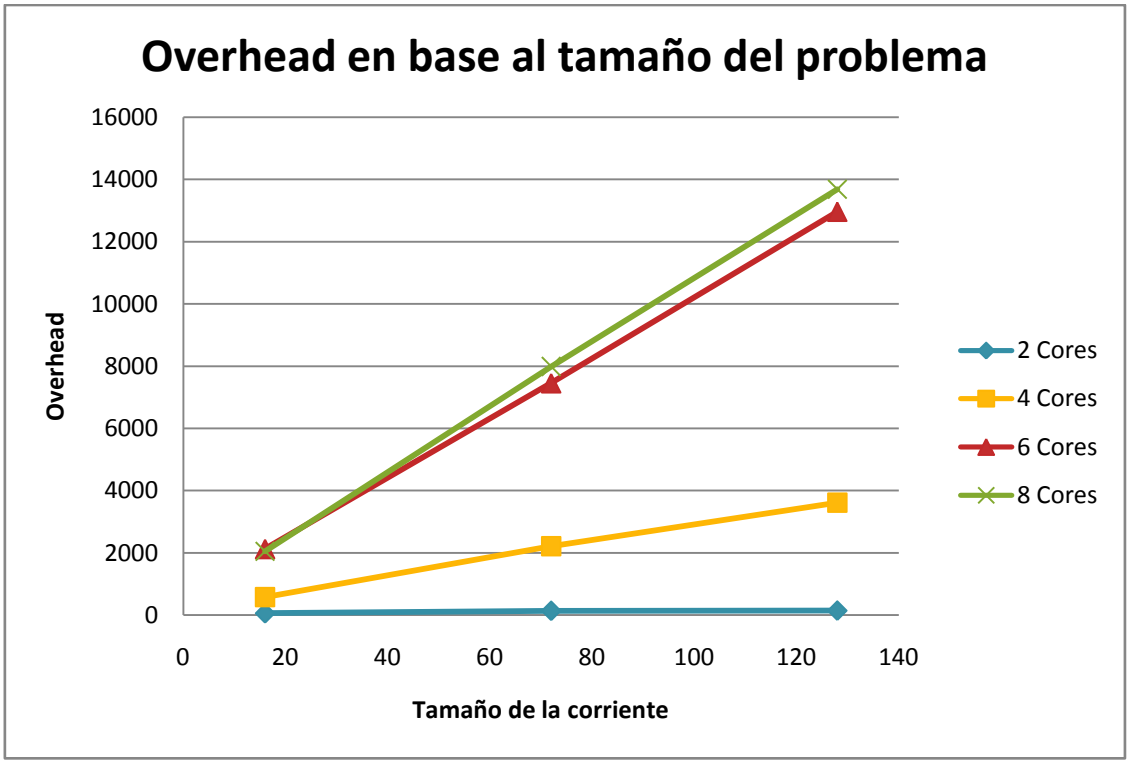

Ilustración 6-13 Gráfica overhead en relación al tamaño del problema, parte 2

Si observamos la relación del overhead con el tamaño del problema, se aprecia el crecimiento lineal de este, también como hasta cuatro cores como el crecimiento del overhead es muy pobre, siendo casi nulo con dos cores, con respecto a los demás. Hay que destacar el excesivo crecimiento del overhead con los seis cores, pero una vez más es la causa del reparto de trabajo el cual no es eficiente para este tamaño de cores.

Ahora pasemos a analizar el overhead con la mayor carga de trabajo.

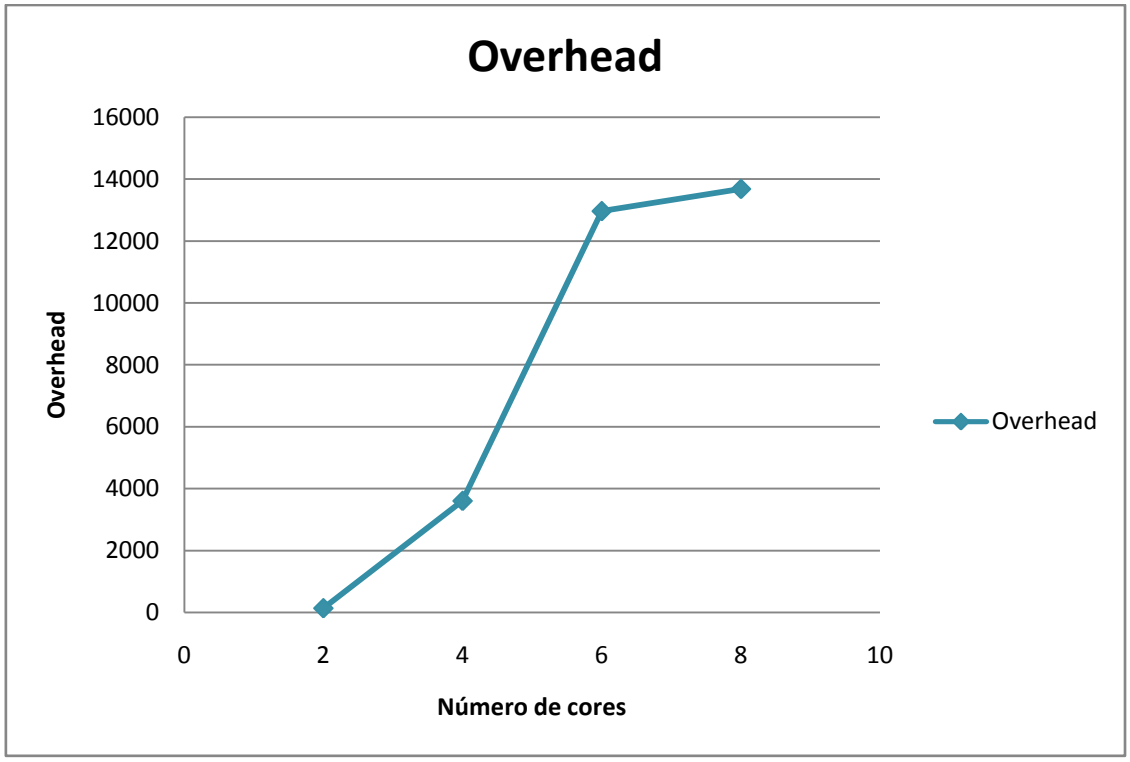

Ilustración 6-14 Gráfica overhead, parte 2

Si observamos esta gráfica pasa lo mismo que en el caso del coste, con el mismo crecimiento exagerado con 6 cores, debido al mismo problema. También se observa que con dos cores el overhead es prácticamente inexistente.

6.3. Otras pruebas realizadas

Se han realizado otras pruebas, en las que se han estudiado los rendimientos con las diferentes estrategias de reparto, la diferencia de escalabilidad del problema en la segunda parte con un bucle de 3 dimensiones y de 1 dimensión tanto en este clúster, como en otro al que hemos podido acceder en la última semana de desarrollo.

A continuación se mostrara la gráfica resultante sobre el estudio de rendimiento de los diferentes repartos estudiados.

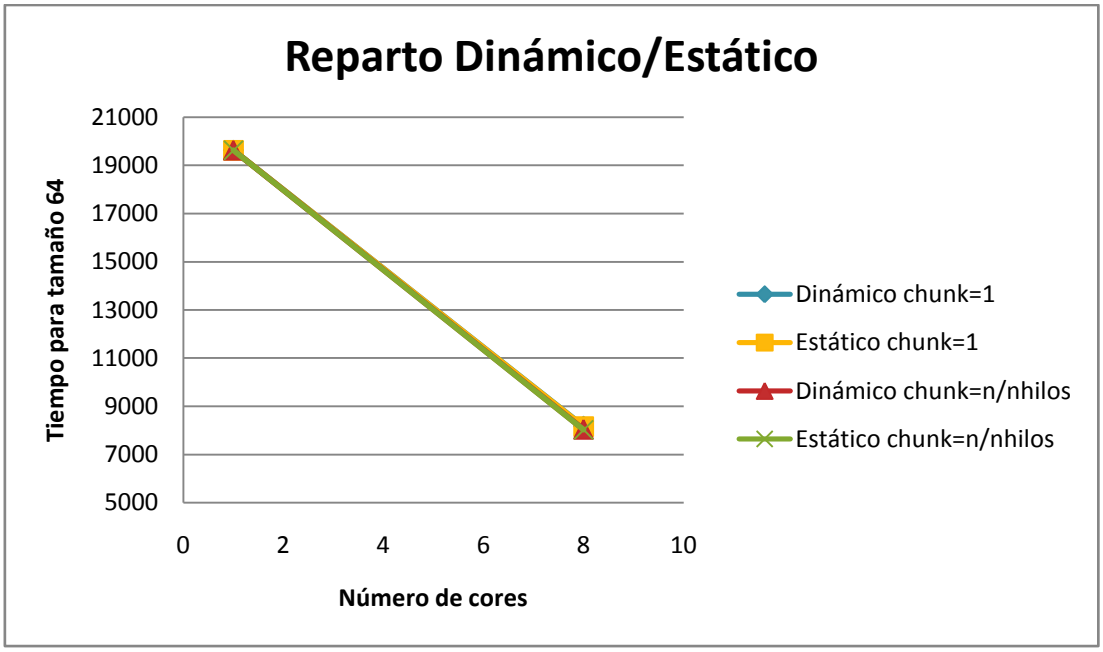

Ilustración 6-15 Gráfica de diferentes repartos

Como se puede apreciar en esta gráfica, todos las estrategias estudiadas son similares en base al rendimiento. Por ello se ha optado por la dinámica con tamaño de reparto uno, ya que esta estrategia evita que los cores estén ociosos y al repartir el trabajo de uno en uno se evita la posible sobre carga de trabajo en determinados cores.

Ahora veamos cómo se comporta la paralelización en la segunda parte con la estrategia de paralelizar en un bucle de tres dimensiones, en vez de con una dimensión, para realizar estos estudios, se ha podido contar con el acceso a un clúster diferente más potente.

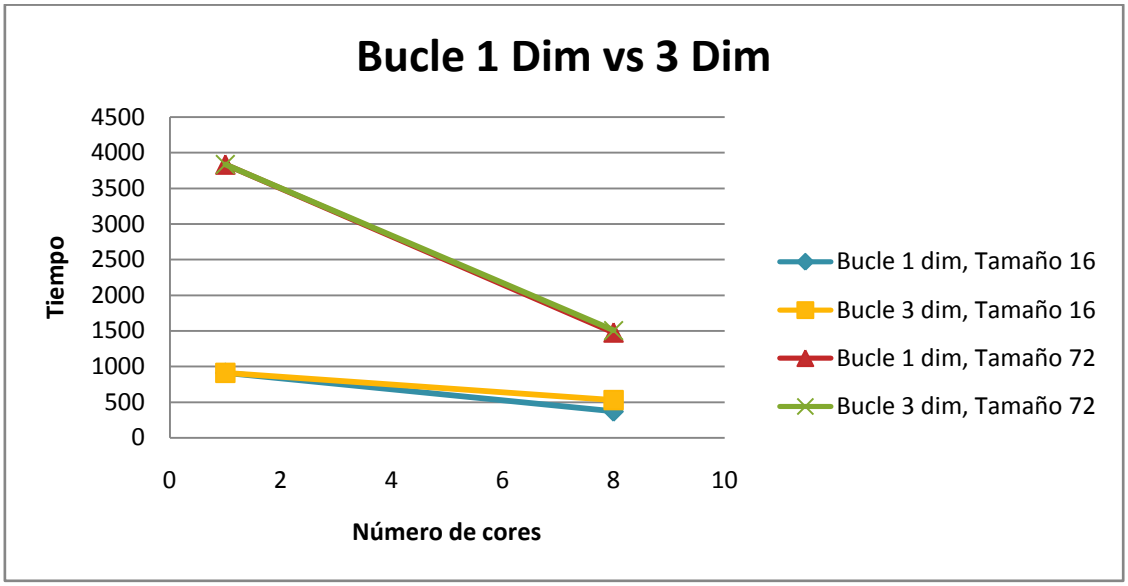

Ilustración 6-16 Gráfica diferencia de rendimientos 1 dimensión vs. 3 dimensiones

Como se puede observar en esta gráfica con una dimensión siempre es más escalable, esto se debe a que con tres dimensiones para ocho cores siempre se quedan cores en espera, ya que en el caso de la matriz con tamaño 16 es de 4x4x1 pudiéndose repartir solo el trabajo entre cuatro cores, habiendo reservado memoria para ocho cores previamente, mientras que con una dimensión seria un bucle de uno hasta ocho, pudiéndose utilizar los ocho cores. Para el tamaño 72, seria de 6x6x2 y en este caso quedándose dos cores inútiles, por eso el rendimiento es más parejo.

A continuación lo estudiaremos para el segundo clúster usando los tamaños de 288 (12x8x3) y de 144(12x4x3).

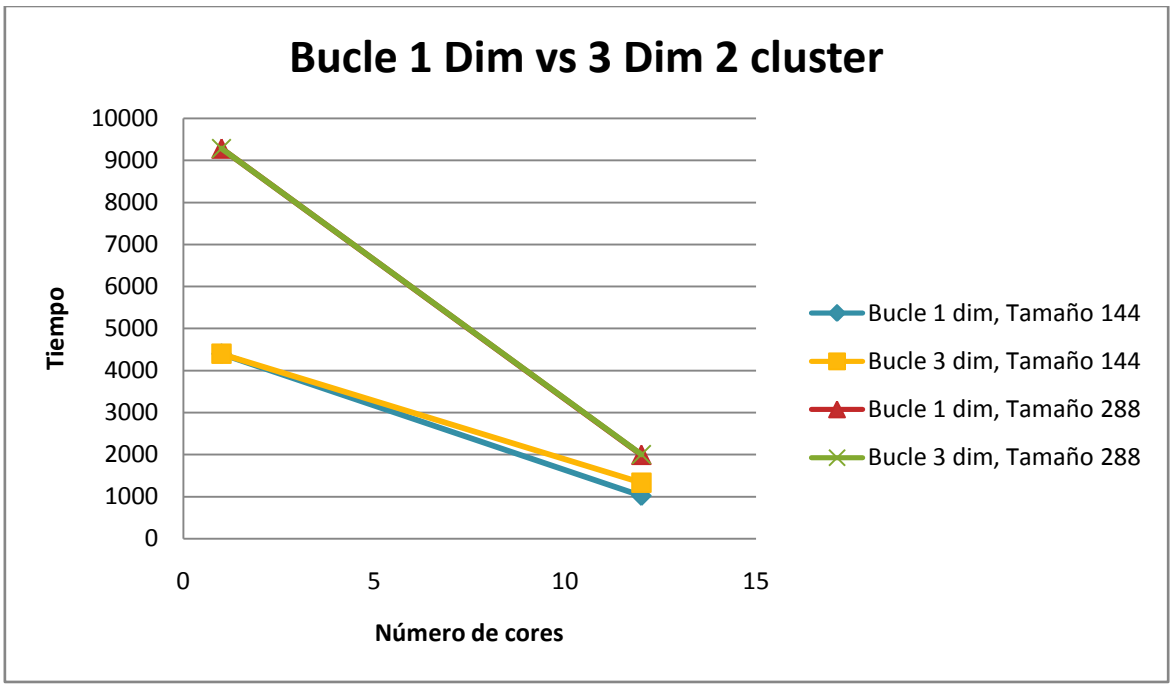

Ilustración 6-17 Gráfica diferencia de rendimientos 1 dimensión vs. 3 dimensiones en el segundo clúster

Como se puede ver en las tablas del anexo D, el speed-up conseguido en este cluster para 12 cores es de 4,64 con un tamaño de 288 frente al 2,65 con un tamaño de 128 conseguido en el otro cluster (anexo C), mejorando la eficiencia en 0,08 puntos.

Hay que decir, que aunque este clúster es más potente y consta de más núcleos, también sufre penalizaciones, las cuales se pueden ver en el anexo E. Pero aun así se demuestra que el programa es bastante escalable y puede llegar a tener mejor rendimiento en un clúster potente y sin penalizaciones de acceso a memoria.

7. Conclusiones

En este trabajo se ha realizado un estudio e implementado una paralelización de tipo fork-join con memoria compartida buscando la optimización del programa en serie. El trabajo se ha dividido en varias fases: una primera fase de análisis y comprensión del código, una segunda parte de paralelización, y una última fase de análisis de rendimientos.

La primera fase a su vez se ha dividido en tres partes. La primera parte ha sido dedicada al aprendizaje del lenguaje de programación Fortran, la segunda entendimiento y análisis del código del programa y la tercera parte análisis de las dependencias en las zonas a paralelizar.

La segunda y tercera fase ha consistido en seguir distintas estrategias de paralelización y asegurarse de la correcto funcionamiento de estas, con el objetivo de obtener el mayor rendimiento del programa.

Tras llevar a cabo el análisis de la paralelización con test que simulan celdas de pocos átomos se han encontrado distintas limitaciones como:

- Problemas de hardware en el sistema donde se han analizado los test.
- Limitaciones por el gran uso de módulos y de variables globales.
- Problemas en el cambio del código, a la hora de modificar subrutinas.
- Problemas en el cambio del código, a la hora de pasar variables de varias dimensiones por subrutinas.
- Problemas de encontrar todas las dependencias, por el gran uso de módulos y de variables globales usadas en ellos.
- Problemas con la manipulación de variables dinámicas en zonas paralelas.
- Limitaciones por tener y depender de actualizaciones en el código en serie proporcionas por el cliente.
- El tamaño de los ficheros a analizar de las funciones de Green, hizo que se tuviera que modificar la escritura, para separarlo en varios ficheros más pequeños para poder verificar que estos cálculos eran correctos.
- El acceso a otro clúster, pero en la última semana de trabajo.

Los resultado obtenidos en los análisis de rendimiento demuestran que:

- La paralelización es efectiva hasta que las limitaciones la hacen poco eficiente.

- Es escalable, y en otra plataforma multicore debería aumentar el rendimiento.

- A pesar de que los resultados obtenidos no son del todo positivos, es necesaria una ampliación del estudio que permita determinar con mayor exactitud si la paralelización es eficaz, evaluándola en otros sistemas multicore.

- La evaluación en otro sistema multicore, ha sido satisfactorio, aumentando en un 50% la paralelización se ha conseguido casi un aumento del 100% aumentando también el trabajo computacional.

Trabajo futuro

Se plantea la posibilidad de hacer una programación paralela hibrida juntando OpenMP con MPI, es decir uso de paso de mensajes junto a la memoria compartida. Teniendo las ventajas de los dos métodos de paralelización.

Anexo A: Fichero de salida Stm.out

```
================================================
stm program for current calculations
CGP Prague  2008
================================================

.....
Welcome to the readstm subroutine.
It reads the stm.inp file with the parameters
of the options you can chose in the program.
You are using a fb2005 sample.
.....
einitial_smp:   0.000000000000000E+000 , efinal_smp:   0.200000000000000
nenergy_smp:           4 , eimag_smp:   5.000000000000000E-002
einitial_tip:  -0.200000000000000      , efinal_tip:   0.000000000000000E+000
nenergy_tip:           4 , eimag_tip:   5.000000000000000E-002
* Cambioamp parameter:  5.165344133333333E-006
* We will read the Hamilt & Overlap files
* The green function of the smp will be calculated from scratch and a file will
be written with this info
* The green function of the tip will be calculated from scratch and a file will
be written with this info
* You have chosen sample orbitals scan
* No orbital contribution will be written
* You have chosen both denominator scan
* You have chosen STM calculations
 * You have chosen old hoppings format
 * The scan is from (   1.86254800000000      ,   1.31692300000000      ,
  9.31144800000000      ) to (  0.000000000000000E+000 ,
 0.000000000000000E+000 ,   9.31144800000000      )
 .....
  Welcome to the subroutine that reads the
    structural properties of the sample.
 .....

 +++ Sample information:

 natoms_smp,natom_smpini,natom_smpend:
        24         21         24
 number of orbitals and ntypes:   9.00000000000000             1
 number of shells:           3
 Lattice vectors of the smp
14.0780806697153       0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000   9.95470631051261       0.000000000000000E+000
 norb_smpini,norb_smpend,norb  181  216   36
.....
    Welcome to read_Ham_Over_omx subroutine
 It reads formated pieces of Hamiltonian from
              the sample.
```

```
.............................................
Reads the Hamiltonian of atom 1  ...
Num. neighbors:          44
Reads the Hamiltonian of atom 2  ...
...
Reads the Hamiltonian of atom 23 ...
Num. neighbors:          46
Reads the Hamiltonian of atom 24 ...
Num. neighbors:          46
Reads the overlaps of atom          1
Reads the overlaps of atom          2
...
Reads the overlaps of atom         23
Reads the overlaps of atom         24

Finishing the read_Ham_Over_omx subroutine for the smp
fermi_level_smp:  -4.17263020000000

.............................................
      Welcome to the green subroutine
   Here the Green function and DOS of the
       sample are calculated or read
.............................................

estep, e0:    0.066667    0.000000

 Assemble non-orthogonal interactions...
Calculating  S^-1/2
K point no.:          1  of          4

 Assemble non-orthogonal interactions...
Calculating  S^-1/2
K point no.:          2  of          4

 Assemble non-orthogonal interactions...
Calculating  S^-1/2
K point no.:          3  of          4

 Assemble non-orthogonal interactions...
Calculating  S^-1/2
K point no.:          4  of          4

 * Green Function and DOS calculated for the smp.
 * Green Function and DOS written in the GreenFunc_smp.out file.

 Exiting green (smp)...
.............................................
  Welcome to the subroutine that reads the
     structural properties of the tip
.............................................

+++ Tip information:
```

```
 natoms_tip,natom_tipini,natom_tipend:
         10         1         1
 number of orbitals and ntypes:   9.00000000000000              1
 number of shells:          3
 Lattice vectors of the tip
18.8972598857892       0.000000000000000E+000  0.000000000000000E+000
 0.000000000000000E+000   18.8972598857892       0.000000000000000E+000
 norb_tipini,norb_tipend,norb    1    9    9
.....................................................
    Welcome to read_Ham_Over_omx subroutine
  It reads formated pieces of Hamiltonian from
                the tip.
 .....................................................
 Reads the Hamiltonian of atom 1  ...
 Num. neighbors:          9
 Reads the Hamiltonian of atom 2  ...
 Num. neighbors:          9
 ...
Reads the Hamiltonian of atom 10 ...
 Num. neighbors:          12
 Reads the overlaps of atom          1
 Reads the overlaps of atom          2
...
 Reads the overlaps of atom         10

 Finishing the read_Ham_Over_omx subroutine for the tip
 fermi_level_tip:  -2.46293160000000
 .....................................................
       Welcome to the green subroutine
    Here the Green function and DOS of the
         tip are calculated or read
 .....................................................
 estep, e0:    0.066667   -0.200000

  Assemble non-orthogonal interactions...
 Calculating  S^-1/2
 K point no.:          1  of          4
...
 Assemble non-orthogonal interactions...
 Calculating  S^-1/2
 K point no.:          4  of          4

  * Green Function and DOS calculated for the tip.
  * Green Function and DOS written in the GreenFunc_tip.out file.

  Exiting green (tip)...
 Calling tip interpolation subroutine...
dos_tip(1,1)          1 -0.200000000000000       2.905952268122799E-002
 dos_tip(1,1)          2 -0.133333333333333       6.039060218430090E-002
 dos_tip(1,1)          3 -6.666666666666669E-002  0.183096640622486
 dos_tip(1,1)          4 -2.775557561562891E-017  0.206461829287869
```

```
Exiting tip interpolation subroutine...
 Calling hoppings reading subroutine...
 Reading tip_sample_hop files...
 atoms          1          1
 Number of interactions in this case:          14
  Short range hopping as a function of distance
 Exiting hoppings reading subroutine...
hbeg:      1   10   19   28
hend:      9   18   27   36

 ---------------------- SCANNING ----------------------
                Using the formulae:
  J=(4*pi*e)/h sum_E(T_12 rho_22 D_r T_21 D_a rho_11)
 ------------------------------------------------------
 ..........................................................
istep:   0 ; jstep:   0 ; kstep:   0
Current calculation data: (  0.000,  0.200) with   4   energy steps: (  1,  4)
Current calculation data: (  0.000,  0.200) with   4   energy steps: (  1,  4)
---- Current no deno    0.47731E-03   1.86255   1.31692   9.31145
---- Current with deno    0.11075E-04   1.86255   1.31692   9.31145
 ..........................................................
istep:   1 ; jstep:   0 ; kstep:   0
Current calculation data: (  0.000,  0.200) with   4   energy steps: (  1,  4)
Current calculation data: (  0.000,  0.200) with   4   energy steps: (  1,  4)
---- Current no deno    0.26745E-03   3.35251   1.31692   9.31145
---- Current with deno    0.12886E-04   3.35251   1.31692   9.31145

..........................................................
..........................................................
Writing the ind_deno=nden(1) file
Writing the ind_deno=deno(2) file
```

Anexo B: Tablas de la Parte 1

Tiempo

Tamaño	Serie	2 Cores	4 Cores	6 Cores	8 cores
4	0,965	0,803	0,604	0,603	0,605
16	94,807	152,408	137,13	137,032	131,568
32	227,653	305,817	270,685	267,635	263,684
64	19623,442	10132,191	8120,557	8114,954	8173,621

Tamaño 64	Speed-up
2 Cores	1,936742211
4 Cores	2,416514286
6 Cores	2,418182777
8 Cores	2,400826023

Tamaño 64	Overhead
2 Cores	640,94
4 Cores	12858,786
6 Cores	29066,282
8 Cores	45765,526

Tamaño 32	Speed-up
2 Cores	0,744409238
4 Cores	0,841025546
6 Cores	0,850609973
8 Cores	0,86335538

Tamaño 32	Overhead
2 Cores	383,981
4 Cores	855,087
6 Cores	1378,157
8 Cores	1881,819

Tamaño 16	Speed-up
2 Cores	0,622060522
4 Cores	0,691365857
6 Cores	0,691860295
8 Cores	0,720593153

Tamaño 16	Overhead
2 Cores	210,009
4 Cores	453,713
6 Cores	727,385
8 Cores	957,737

Tamaño 4	Speed-up
2 Cores	1,201743462
4 Cores	1,597682119
6 Cores	1,600331675
8 Cores	1,595041322

Tamaño 4	Overhead
2 Cores	0,641
4 Cores	1,451
6 Cores	2,653
8 Cores	3,875

Tamaño 64	Eficiencia
2 Cores	0,968371106
4 Cores	0,604128571
6 Cores	0,403030463
8 Cores	0,300103253

Tamaño 64	coste
2 Cores	20264,382
4 Cores	32482,228
6 Cores	48689,724
8 Cores	65388,968

Speed-up

Tamaño	Core 2	Core 4	Core 6	Core 8
4	1,201743462	1,597682119	1,600331675	1,595041322
16	0,622060522	0,691365857	0,691860295	0,720593153
32	0,744409238	0,841025546	0,850609973	0,86335538
64	1,936742211	2,416514286	2,418182777	2,400826023

Overhead

Tamaño	2 Cores	4 Cores	6 Cores	8 Cores
4	0,641	1,451	2,653	3,875
16	210,009	453,713	727,385	957,737
32	383,981	855,087	1378,157	1881,819
64	640,94	12858,786	29066,282	45765,526

Anexo C: Tablas de la Parte 2

Tiempo

Tamaño	Serie	2 Cores	4 Cores	6 Cores	8 cores
16	909,617	482,635	372,115	505,643	369,278
72	3831,497	1983,34	1511,006	1880,494	1477,088
128	6790,091	3465,678	2599,247	3292,315	2558,657

Tamaño 128	Speed-up
2 Cores	1,959238856
4 Cores	2,612330033
6 Cores	2,06240624
8 Cores	2,65377149

Tamaño 128	Overhead
2 Cores	141,265
4 Cores	3606,897
6 Cores	12963,799
8 Cores	13679,165

Tamaño 72	Speed-up
2 Cores	1,931840733
4 Cores	2,535725867
6 Cores	2,037494935
8 Cores	2,593953102

Tamaño 72	Overhead
2 Cores	135,183
4 Cores	2212,527
6 Cores	7451,467
8 Cores	7985,207

Tamaño 16	Speed-up
2 Cores	1,884689258
4 Cores	2,444451312
6 Cores	1,798931262
8 Cores	2,463230953

Tamaño 16	Overhead
2 Cores	55,653
4 Cores	578,843
6 Cores	2124,241
8 Cores	2044,607

Tamaño 128	Eficiencia
2 Cores	0,979619428
4 Cores	0,653082508
6 Cores	0,343734373
8 Cores	0,331721436

Tamaño 128	coste
2 Cores	6931,356
4 Cores	10396,988
6 Cores	19753,89
8 Cores	20469,256

Speed-up

Tamaño	Core 2	Core 4	Core 6	Core 8
16	1,884689258	2,4444513	1,798931262	2,463230953
72	1,931840733	2,5357259	2,037494935	2,593953102
128	1,959238856	2,61233	2,06240624	2,65377149

Overhead

Tamaño	2 Cores	4 Cores	6 Cores	8 Cores
16	55,653	578,843	2124,241	2044,607
72	135,183	2212,527	7451,467	7985,207
128	141,265	3606,897	12963,799	13679,165

Anexo D: Tablas de otras pruebas

Parte 1 en Dinámico

Tamaño	Serie	8 cores Chunk=1	8 cores Chunk= n/numHilos
64	19623,442	8173,621	8039

Parte 1 en estático

Tamaño	Serie	8 cores Chunk=1	8 cores Chunk= n/numHilos
64	19623,442	8158,973	8027,076

Parte 2 con 8 cores

Tamaño	serie	bucle 1 dim	bucle 3 dimensiones
16	909,617	369,278	528,472
72	3831,497	1477,088	1505,779

2º Cluster 12 cores

Tamaño	Serie	bucle 1 dim	bucle 3 dimensiones
144	4398,373	1020,328	1332,178
288	9281,758	1998,747	2007,889

Tamaño 144	Speed-up
1 Dim	4,31074419
3 Dim	3,30164062

Tamaño 144	Overhead
1 Dim	7845,563
3 Dim	10911,758

Tamaño 288	Speed-up
1 Dim	4,64378833
3 Dim	4,62264498

Tamaño 288	Overhead
1 Dim	14703,206
3 Dim	21977,075

Tamaño 144	Eficiencia
1 Dim	0,35922868
3 Dim	0,27513672

Tamaño 144	coste
1 Dim	12243,936
3 Dim	15986,136

Tamaño 288	Eficiencia
1 Dim	0,38698236
3 Dim	0,38522041

Tamaño 288	coste
1 Dim	23984,964
3 Dim	24094,668

Anexo E: Tablas de conflictos

Tabla de conflictos entre cores en el cluster 1:

	0	1	2	3	4	5	6	7
0	X	259m4.249s	168m52.191s	168m47.327s	291m9.357s	259m7.413s	171m2.938s	170m54.605s
1		X	168m54.122s	168m53.005s	258m24.920s	291m8.858s	169m33.166s	169m46.166s
2		---	X	259m1.679s	170m34.849s	169m30.201s	291m31.740s	258m44.779s
3		---	---	X	169m7.297s	169m23.584s	259m29.079s	291m3.073s
4		---	---	---	X	259m7.957s	168m45.166s	168m48.006s
5		---	---	---	---	X	168m54.452s	168m53.100s
6		---	---	---	---	---	X	259m28.521s
7		---	---	---	---	---	---	X

Tabla de conflictos entre cores en el cluster 2

	0	1	2	3	4	5	6	7	8	9	10	11
0	X	50	50	50	50	50	33	33	33	33	33	33
1	50	X	50	50	50	50	33	33	33	33	33	33
2	50	50	X	50	50	50	33	33	33	33	33	33
3	50	50	50	X	50	50	33	33	33	33	33	33
4	50	50	50	50	X	50	33	33	33	33	33	33
5	50	50	50	50	50	X	33	33	33	33	33	33
6	33	33	33	33	33	33	X	50	50	50	50	50
7	33	33	33	33	33	33	50	X	50	50	50	50
8	33	33	33	33	33	33	50	50	X	50	50	50
9	33	33	33	33	33	33	50	50	50	X	50	50
10	33	33	33	33	33	33	50	50	50	50	X	50
11	33	33	33	33	33	33	50	50	50	50	50	X

Referencias

[1] *Introduction to Scanning Tunneling Microscopy Second Edition.* C. Julian Chen, 2007 URL: http://www.columbia.edu/~jcc2161/documents/STM_2ed.pdf.

[2] Estudio *teórico del Microscopio de Efecto Túnel con métodos de primeros principios.* José Manuel Blanco Ramos, 2004.

[3] https://es.wikipedia.org/wiki/Efecto_t%C3%BAnel

[4] http://www.capri-school.eu/capri09/lectures/Kamenev_Capri09.pdf

[5] *Intrusive STM imaging*. Nicolas Boulanger-Lewandowski and Alain Rochefort *Department de genie physique and Regroupement queb écois sur les matériaux de pointe (RQMP),École Polytechnique de Montreal, Montreal, Quebec H3C 3A7, Canada* 2010

[6] https://es.wikipedia.org/wiki/Fortran

[7] https://es.wikipedia.org/wiki/OpenMP

[8] *Strict Fork-Join Parallelism.* Pablo Halpern, 2012 URL: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3409.pdf

[9] https://software.intel.com/en-us/articles/using-kmp-affinity-to-create-openmp-thread-mapping-to-os-proc-ids

[10] https://es.wikipedia.org/wiki/Dependencia_de_datos

[11] https://es.wikipedia.org/wiki/MIMD

[12] https://es.wikipedia.org/wiki/Ley_de_Amdahl